Ingeniería de Servidores

Ejercicios

Ismael Sallami Moreno

DGIIADE

Índice general

1 Ejercicios Tema 3

1.1. Conversión de unidades de KiB a otras y viceversa

En informática, las unidades de medida de almacenamiento suelen expresarse en múltiplos binarios (potencias de 1024). A continuación se describe cómo convertir desde **KiB** (Kibibytes) a otras unidades y viceversa.

- **1 KiB (Kibibyte)** = 1024 bytes
- **1 MiB (Mebibyte)** = 1024 KiB = 1.048.576 bytes
- **1 GiB (Gibibyte)** = 1024 MiB = 1.073.741.824 bytes
- **1 TiB (Tebibyte)** = 1024 GiB = 1.099.511.627.776 bytes

1.1.1. Fórmulas para conversión

- Para pasar de **KiB a bytes**:

$$\text{bytes} = \text{KiB} \times 1024$$

- Para pasar de **bytes a KiB**:

$$\text{KiB} = \frac{\text{bytes}}{1024}$$

- Para pasar de **KiB a MiB**:

$$\text{MiB} = \frac{\text{KiB}}{1024}$$

- Para pasar de **MiB a KiB**:

$$\text{KiB} = \text{MiB} \times 1024$$

- Para pasar de **KiB a GiB**:

$$\text{GiB} = \frac{\text{KiB}}{1024 \times 1024}$$

- Para pasar de **GiB a KiB**:

$$\text{KiB} = \text{GiB} \times 1024 \times 1024$$

1.1.2. Ejemplo de conversiones rápidas

- **2048 KiB** = 2048 × 1024 = 2.097.152 bytes
- **2048 KiB** = 2048 / 1024 = 2 MiB
- **5242880 bytes** = 5242880 / 1024 = 5120 KiB
- **3 GiB** = 3 × 1024 × 1024 = 3.145.728 KiB

Estas conversiones permiten ajustar los datos a la unidad más adecuada según la magnitud de la información o el análisis que se esté realizando.

1.2. PROBLEMA 3.1

En un sistema Linux se ha ejecutado la orden `uptime` tres veces en momentos diferentes. El resultado, de forma resumida, es el siguiente:

```
... load average: 6.85, 7.37, 7.83
... load average: 8.50, 10.93, 8.61
... load average: 37.34, 9.47, 3.30
```

Se pide determinar si la carga del sistema crece, decrece, se mantiene estacionaria o bien no se puede decidir sobre ello.

Explicación:

El `load average` representa el número medio de procesos en ejecución o esperando ser ejecutados (en estado runnable o I/O blocked) durante los últimos 1, 5 y 15 minutos. Constituye un buen indicador de la carga de trabajo del sistema.

Al analizar las tres ejecuciones observamos lo siguiente:

- En la primera medida, los valores son relativamente cercanos entre sí (en torno a 7–8).
- En la segunda medida, la carga se incrementa notablemente, especialmente en la media de los últimos 5 minutos (10.93).
- En la tercera medida, se produce un pico muy elevado en el promedio de 1 minuto (37.34), pero los promedios de 5 y 15 minutos descienden respecto a la segunda medida.

Estas variaciones muestran fluctuaciones significativas en la carga del sistema, sin seguir una tendencia uniforme o constante. Dado que no se observa un patrón claro de crecimiento o decrecimiento continuo en los valores, no es posible determinar si la carga del sistema está aumentando, disminuyendo o permaneciendo estable.

Por tanto, la respuesta correcta es que no se puede decidir sobre la evolución de la carga a partir de los datos disponibles.

1.3. PROBLEMA 3.2

En un sistema Linux se ha ejecutado la siguiente orden:

```
$ time quicksort
real 0m40.2s
user 0m17.1s
sys  0m3.2s
```

Indique si el sistema está soportando mucha o poca carga. Razone la respuesta.

Explicación:

El comando `time` nos muestra tres valores principales:

- **real**: tiempo total transcurrido desde que comenzó la ejecución del comando hasta que finalizó.
- **user**: tiempo que la CPU dedicó a ejecutar instrucciones en modo usuario.
- **sys**: tiempo que la CPU dedicó a ejecutar instrucciones en modo núcleo del sistema.

En este caso, el tiempo real (40.2 segundos) es considerablemente superior a la suma de los tiempos de usuario y sistema ($17.1 + 3.2 = 20.3$ segundos). Esto sugiere que el proceso ha pasado más de la mitad del tiempo total esperando recursos, probablemente debido a la contención de recursos del sistema o al uso intensivo de operaciones de entrada/salida.

Aunque podría deberse a que el propio programa quicksort realiza muchas operaciones de E/S (lo que también explicaría la diferencia entre el tiempo real y el tiempo de CPU), es más probable que la diferencia significativa se deba a que el sistema tiene una carga elevada. Esto provoca que el proceso deba esperar más tiempo para que la CPU y otros recursos estén disponibles, retrasando así su finalización.

En consecuencia, se concluye que el sistema soporta una carga considerable.

1.4. PROBLEMA 3.3

Se sabe que la sobrecarga (overhead) de CPU de un monitor software en un determinado servidor es del 4 %. Si el monitor se activa cada 2 segundos, ¿cuánto tiempo tarda el monitor en ejecutarse por cada activación?

Explicación:

La sobrecarga que introduce el monitor de actividad está definida como el porcentaje de tiempo de CPU que consume respecto al tiempo total entre dos activaciones consecutivas. La fórmula, según se indica en los apuntes, es:

$$\text{Sobrecarga}\ (\%) = \left(\frac{\text{tiempo de CPU que usa el monitor por activación}}{\text{periodo de activación}}\right) \times 100$$

Despejando el tiempo de CPU que usa el monitor por activación:

$$\text{tiempo de CPU que usa el monitor por activación} = \frac{\text{Sobrecarga (\%)} \times \text{periodo de activación}}{100}$$

Sustituyendo los datos del problema (sobrecarga = 4 %, periodo de activación = 2 s):

$$\text{tiempo de CPU que usa el monitor por activación} = \frac{4 \times 2}{100} = 0.08\,s = 80\,ms$$

Por lo tanto, el monitor tarda 80 milisegundos en ejecutarse por cada activación.

1.5. PROBLEMA 3.4

A continuación se muestra el resultado obtenido tras ejecutar la orden `top` en un sistema informático que emplea Linux como sistema operativo:

```
2:52pm up 17 days, 3:41, 1 user, load average: 0.15, 0.27, 0.32
54 processes: 51 sleeping, 3 running, 0 zombie, 0 stopped
%Cpu(s): 23.8% user, 14.0% system, 0.0% nice, 17.0% idle, 45.2% wa
Mem:  257124K av, 253052K used,  4072K free, 8960K shrd, 182972K buff
Swap: 261496K av,  21396K used,240100K free, 26344K cached
PID USER PRI NI VIRT RSS SHARE STAT LC %CPU %MEM TIME COMMAND
807 joan 0 0 5708 5708 532 R N 0 23.0 2.2 6:16 p_exec
809 joan 0 0 5708 5708 532 R N 0 14.0 2.2 3:42 p_exec
185 tomi 0 0 824 824 632 R 0 0.5 0.3 0:00 top
201 xp 0 0 1272 1208 644 S 0 0.1 0.4 5:49 xp_stat
1 root 0 0 60 56 36 S 0 0.0 0.0 0:03 init
2 root 0 0 0 0 0 SW 0 0.0 0.0 0:13 kflushd
7 root 0 0 0 0 0 SW 0 0.0 0.0 0:00 nfsiod
194 root 0 0 72 4 4 S 0 0.0 0.0 0:00 migetty
195 root 0 0 68 0 0 SW 0 0.0 0.0 0:00 migetty
179 root 0 0 532 312 236 S 0 0.0 0.1 0:00 sndmail
```

Se pide responder a las siguientes preguntas:

a) ¿Cuánta memoria física tiene la máquina?

b) ¿Qué porcentaje de la memoria física está marcada como usada según el monitor?

c) ¿Cuál es la utilización media del procesador?

d) ¿Cómo es la evolución de la carga media del sistema, ascendente o descendente?

e) ¿Cuánta memoria física ocupa el monitor?

Explicación:

a) La memoria física disponible en la máquina es de 257124 KiB, como se indica en la línea de memoria:

```
Mem:  257124K av, 253052K used,  4072K free, ...
```

b) El porcentaje de memoria física utilizada se calcula como:

$$\frac{253052}{257124} \times 100 \approx 98.4\,\%$$

c) La utilización media del procesador corresponde a la suma de los tiempos de usuario y de sistema:

$$23.8\,\% + 14.0\,\% = 37.8\,\%$$

d) La evolución de la carga media es descendente, ya que los promedios de carga son:

```
0.15 (1 min), 0.27 (5 min), 0.32 (15 min)
```

El valor más reciente (1 min) es menor que los promedios a más largo plazo, indicando que la carga está disminuyendo.

e) La memoria que ocupa el monitor `top` es de 824 KiB, como aparece en la línea del proceso `top`:

```
185 tomi 0 0 824 824 632 R 0 0.5 0.3 0:00 top
```

- `VIRT`: memoria virtual.
- `RSS`: (Resident Set Size) memoria física.

En cuanto a la respuesta de la pregunta nos debemos de quedar con la física. La unidad siempre es en KiB (Kibibyte).

Por lo tanto, las respuestas finales son: a) 257124 KiB.

b) 98.4 %.

c) 37.8 %.

d) Descendente.

e) 824 KiB.

1.6. PROBLEMA 3.5

Considere las órdenes siguientes ejecutadas en un sistema Linux:

```
$ time simulador_original
real 0m24.2s
user 0m15.1s
sys  0m1.6s

$ time simulador_mejorado
real 0m32.8s
user 0m10.7s
sys  0m2.1s
```

Se pide:

1. ¿Cuál es el tiempo de ejecución de ambos simuladores?
2. Calcule, si es el caso, la mejora en el tiempo de ejecución del simulador mejorado respecto del original.

Explicación:

1. El tiempo efectivo de ejecución de un programa es la suma de los tiempos de CPU en modo usuario y en modo sistema:

- Para el **simulador original**:

$$\text{user} + \text{sys} = 15.1\,s + 1.6\,s = 16.7\,s$$

- Para el **simulador mejorado**:

$$\text{user} + \text{sys} = 10.7\,s + 2.1\,s = 12.8\,s$$

Aunque el tiempo **real** o de reloj es mayor en el caso del simulador mejorado (32.8 s frente a 24.2 s), este valor también incluye el tiempo que el proceso ha pasado esperando a la disponibilidad de la CPU u otros recursos del sistema (tiempo de espera por carga del sistema). Por tanto, la comparación más relevante es la de los tiempos de CPU (`user + sys`), que miden la cantidad de trabajo efectivo que realizó la CPU para cada simulador.

2. La mejora en el tiempo de ejecución se calcula comparando los tiempos de CPU:

$$\text{Mejora} = \frac{\text{tiempo original}}{\text{tiempo mejorado}} = \frac{16.7}{12.8} \approx 1.3$$

Esto significa que el **simulador mejorado** es aproximadamente **1,3 veces más rápido** que el simulador original en cuanto a la carga real de trabajo de la CPU.

Aunque los tiempos reales puedan inducir a error, estos reflejan la carga del sistema en ese momento y no el rendimiento puro del programa. Por eso, la comparación justa debe realizarse con los tiempos de CPU (`user + sys`).

1.7. PROBLEMA 3.6

El monitor `sar` (system activity reporter) de un computador se activa cada 15 minutos y tarda 750 ms en ejecutarse por cada activación. Se pide:

- Calcular la sobrecarga que genera este monitor sobre el sistema informático.
- Si la información generada en cada activación ocupa 8192 bytes, ¿la monitorización de cuántos días completos se pueden almacenar en el directorio `/var/log/sysstat` si se dispone únicamente de 200 MiB de capacidad libre?

Explicación:

1.7.1. Cálculo de la sobrecarga del monitor

La **sobrecarga** (overhead) se define como el porcentaje de tiempo que consume el monitor en relación con el tiempo total de un periodo de activación. Según la fórmula vista en los materiales:

$$\text{Sobrecarga } (\%) = \left(\frac{\text{tiempo de CPU usado por el monitor por activación}}{\text{periodo de activación}} \right) \times 100$$

Sustituimos los datos:

- **tiempo de CPU por activación**: 750 ms = 0,75 s
- **periodo de activación**: 15 minutos = 900 s

$$\text{Sobrecarga } (\%) = \left(\frac{0.75}{900} \right) \times 100 \approx 0.083\,\%$$

Por tanto, la sobrecarga generada por el monitor es del **0,083 %**.

1.7.2. Cálculo del número de días completos que se pueden almacenar

Cada activación del monitor genera **8192 bytes** de información. Para saber cuántos días completos se pueden almacenar, calculamos primero cuántas activaciones hay en un día:

- **Número de activaciones por día**:

$$24\,\text{h/día} \times \left(\frac{60\,\text{min}}{15\,\text{min}} \right) = 24 \times 4 = 96\,\text{activaciones/día}$$

- **Tamaño de información generado por día**:

$$8192\,\text{bytes/activación} \times 96\,\text{activaciones} = 786432\,\text{bytes/día} \approx 768\,\text{KiB/día}$$

La capacidad de almacenamiento libre disponible es de **200 MiB = 209715200 bytes**.

- **Número de días completos almacenables**:

$$\frac{209715200}{786432} \approx 266\,\text{días completos}$$

Por lo tanto, se pueden almacenar los datos generados por el monitor `sar` durante **266 días completos** en el espacio disponible.

Nota: La conversión de MiB (Mebibytes) a bytes se realiza usando la base binaria (1024) y la siguiente fórmula:

$$1\,\text{MiB} = 1024 \times 1024\,\text{bytes} = 1.048.576\,\text{bytes}$$

Por ejemplo:

- **1 MiB** = 1.048.576 bytes
- **200 MiB** = 200 × 1.048.576 = 209.715.200 bytes

En general, si tienes un valor en MiB y quieres pasarlo a bytes:

$$\text{bytes} = \text{MiB} \times 1.048.576$$

1.8. PROBLEMA 3.7

El día 8 de octubre se ha ejecutado la siguiente orden en un sistema Linux:

```
% ls /var/log/sysstat
-rw-r--r-- 1 root root 3049952 Oct 6 23:50 sa06
-rw-r--r-- 1 root root 3049952 Oct 7 23:50 sa07
-rw-r--r-- 1 root root 2372184 Oct 8 18:40 sa08
```

Suponiendo que la primera muestra se toma a las 0:00 de cada día y que `sadc` se ejecuta con un tiempo de muestreo constante, se pide determinar:

- ¿Cada cuánto tiempo se activa el monitor `sar`?
- ¿Cuál es la anchura de entrada del monitor?

Explicación:

1.8.1. Frecuencia de activación del monitor `sar`

Los ficheros generados por `sar` tienen marcas de tiempo que indican que la última activación del día se produce a las 23:50. Como la primera activación es a las 0:00 y el tiempo de muestreo es constante, deducimos que las activaciones ocurren cada **10 minutos**.

Esto se debe a que en un día hay 24 horas × 6 muestras por hora (cada 10 min) = 144 muestras, y la última activación es justo a las 23:50.

1.8.2. Anchura de entrada del monitor

La anchura de entrada se calcula como la cantidad de bytes que ocupa cada muestra o activación del monitor. Usamos los datos de los ficheros `sa06` y `sa07` (que representan días completos con el mismo patrón):

- Tamaño del fichero de un día: **3.049.952 bytes**
- Número de activaciones al día:

$$24\,\text{horas} \times \left(\frac{60\,\text{min}}{10\,\text{min}}\right) = 24 \times 6 = 144\,\text{activaciones/día}$$

- Anchura de entrada de cada activación:

$$\text{Anchura de entrada (bytes)} = \frac{3.049.952}{144} \approx 21.180\,\text{bytes} \approx 20,7\,\text{KiB o } 21,2\,\text{KB}$$

Por lo tanto:

- El monitor `sar` se activa cada **10 minutos**.
- La anchura de entrada por activación es aproximadamente **20,7 KiB** o **21,2 KB**.

1.9. PROBLEMA 3.8

Indique el resultado que produce la ejecución de las siguientes órdenes sobre un sistema Linux con el monitor `sar` instalado:

```
1. sar
2. sar -A
3. sar -u 1 30
4. sar -uB -f /var/log/sysstat/08
5. sar -d -s 12:30:00 -e 18:15:00 -f /var/log/sysstat/08
```

Explicación de los resultados:

1. `sar` Muestra la utilización global del procesador (CPU) durante el día actual. Presenta estadísticas como el porcentaje de tiempo de usuario, sistema, espera de E/S, y tiempo inactivo.

2. `sar -A` Genera un informe completo con toda la información recogida durante el día actual. Incluye estadísticas de CPU, memoria, dispositivos de almacenamiento, red, entre otras.
3. `sar -u 1 30` Proporciona información sobre la utilización actual del procesador (`-u`), tomando 30 mediciones con un periodo de 1 segundo entre ellas. Es útil para observar la evolución en tiempo real.
4. `sar -uB -f /var/log/sysstat/08` Extrae del fichero histórico del día 8 (`/var/log/sysstat/08`) estadísticas combinadas de utilización del procesador (`-u`) y de paginación de la memoria virtual (`-B`).
5. `sar -d -s 12:30:00 -e 18:15:00 -f /var/log/sysstat/08` Muestra estadísticas de las transferencias de disco (`-d`) realizadas entre las 12:30 y las 18:15 horas del día 8, utilizando la información almacenada en el fichero histórico correspondiente.

1.9.1. Resumen final

1. Utilización del procesador durante el día actual.
2. Toda la información recogida durante el día actual.
3. Utilización actual del procesador: 30 medidas tomadas con un período de un segundo.
4. Utilización del procesador y paginación de la memoria virtual durante el día 8 del mes.
5. Transferencias de disco desde las 12:30 hasta las 18:15 horas del día 8 del mes.

1.10. PROBLEMA 3.9

Después de instrumentar un programa con la herramienta `gprof`, el resultado obtenido ha sido el siguiente:

```
Flat profile:
Each sample counts as 0.01 seconds.

%time  cumulative  self    self    calls  self  s/call  total s/call  name
59.36     27.72    27.72    3      9.24   14.39         reduce
33.08     43.17    15.45    6      2.57    2.57         invierte
 7.56     46.70     3.53    2      1.76    1.76         calcula
```

El grafo de dependencias muestra que el procedimiento `invierte()` es llamado desde el procedimiento `reduce()`.

Se pide:

1. ¿Cuál es el procedimiento cuyo código propio sería más conveniente optimizar?
2. Si el código propio de `reduce()` se sustituye por otro tres veces más rápido, ¿cuánto tiempo tardará en ejecutarse el programa?
3. Si el procedimiento `invierte()` se sustituye por una nueva versión cuatro veces más rápida, ¿qué mejora se obtendrá en el tiempo de ejecución?

4. Calcule cuál es la ganancia en velocidad máxima que se podría conseguir en el tiempo de ejecución mediante la optimización del código del procedimiento `invierte()`.

Explicación:

1.10.1. 1. Procedimiento prioritario para optimizar

El procedimiento `reduce()` consume el mayor porcentaje de tiempo de CPU (59.36 %), y su tiempo propio (27.72 s) es significativamente superior al de `invierte()` (15.45 s) o `calcula()` (3.53 s). Por tanto, el procedimiento **`reduce()`** es el candidato principal a optimizar para reducir el tiempo de ejecución global.

1.10.2. 2. Tiempo de ejecución si `reduce()` se hace tres veces más rápido

El tiempo propio de `reduce()` es 27.72 s. Si se reduce a la tercera parte:

$$27.72\,s/3 = 9.24\,s$$

El tiempo de ejecución de las demás funciones (`invierte()` y `calcula()`) permanece constante:

$$15.45\,s\ (\text{invierte}) + 3.53\,s\ (\text{calcula}) = 18.98\,s$$

Por tanto, el nuevo tiempo total del programa sería:

$$9.24\,s\ (\text{reduce optimizado}) + 18.98\,s = 28.22\,s$$

1.10.3. 3. Mejora si `invierte()` se hace cuatro veces más rápido

El tiempo propio de `invierte()` es 15.45 s. Si se reduce a la cuarta parte:

$$15.45\,s/4 = 3.86\,s$$

El tiempo de ejecución de las demás funciones (`reduce()` y `calcula()`) no cambia:

$$27.72\,s\ (\text{reduce}) + 3.53\,s\ (\text{calcula}) = 31.25\,s$$

El nuevo tiempo de ejecución sería:

$$3.86\,s\ (\text{invierte optimizado}) + 31.25\,s = 35.11\,s$$

La mejora se calcula comparando con el tiempo original:

$$46.7\,s/35.11\,s \approx 1.33$$

Por lo tanto, el programa se ejecutaría **1,33 veces más rápido**.

1.10.4. 4. Ganancia máxima posible optimizando `invierte()`

La ganancia máxima se obtiene al suponer que el tiempo propio de `invierte()` se pudiera reducir a cero (en la práctica, imposible, pero teóricamente interesante). El tiempo del programa sería entonces:

$$46.7\,s - 15.45\,s = 31.25\,s$$

La ganancia máxima en velocidad sería:

$$46.7/31.25 \approx 1.49$$

Por tanto, la ganancia máxima posible optimizando solo `invierte()` es de **1,49 veces**.

Respuestas finales:

1. El procedimiento más conveniente para optimizar es `reduce()`.
2. El programa se ejecutaría en 28,22 segundos.
3. Se obtendría una mejora de 1,33 veces en la ejecución.
4. La ganancia máxima que se podría conseguir con `invierte()` sería 1,49 veces.

1.11. PROBLEMA 3.10

Un informático desea evaluar el rendimiento de un computador por medio del benchmark SPEC CPU 2017. Una vez compilados todos los programas del paquete y lanzado su ejecución, monitoriza el sistema con la orden:

```
vmstat 1 5
```

El resultado de las medidas de este monitor es el siguiente:

```
procs -- memory- swap-- --io  system -- - cpu-
r  b  swpd  free   buff  cache     si  so  bi  bo  in   cs   us  sy  id  wa
0  0    8   14916  92292 833828    0   0   0   3   0    7    3   1  96   0
1  0    8   14916  92292 833828    0   0   0   0  1022  40   100 0   0    0
3  0    8   14916  92292 833828    2   1   16  3  1016  34   99  1   0    0
1  0    8   14916  92292 833828    0   4   0   8  1035  36   98  2   0    0
2  0    8   14916  92292 833828    1   5   4   28 1035  36   99  1   0    0
```

Se pide determinar si, a la vista de estos datos, los resultados obtenidos en la prueba de evaluación serán correctos o no, y justificar la respuesta.

Explicación:

El comando `vmstat` muestra estadísticas clave del sistema, incluyendo la actividad de la memoria virtual y los intercambios de memoria entre la RAM y el disco (campos `si` y `so`, "swapped in" y "swapped out", respectivamente).

En este caso, los valores `si` (swap in) y `so` (swap out) muestran actividad de intercambio en varias de las muestras:

```
3ª línea: si=2, so=1
4ª línea: si=0, so=4
5ª línea: si=1, so=5
```

Esto significa que el sistema está usando el espacio de intercambio (swap) del disco, lo que indica que la RAM no es suficiente para soportar toda la carga de trabajo. Sin embargo, los programas del benchmark SPEC CPU 2017 están diseñados para realizar pruebas de CPU puras, sin involucrar E/S con el disco ni operaciones de intercambio de memoria.

Por tanto, la actividad de swap que aparece en estas medidas es un indicio de que **los resultados obtenidos por el benchmark no serán correctos**. El sistema operativo ha tenido que intercambiar páginas de memoria con el disco, lo cual introduce tiempos de espera adicionales y hace que las mediciones de rendimiento de CPU estén afectadas por factores ajenos a la propia CPU.

Respuesta final:

El sistema presenta actividad de intercambio con el disco (`swap in` y `swap out`), que no debería ocurrir durante la ejecución de los programas del benchmark. Por tanto, los resultados obtenidos **no serán correctos** porque estarán contaminados por la carga de E/S y la falta de memoria disponible.

1.12. PROBLEMA 3.11

La monitorización de un programa de dibujo en tres dimensiones mediante la herramienta gprof ha proporcionado la siguiente información (por errores en la transmisión hay valores que no están disponibles):

Flat profile:

% time | cumulative seconds | self seconds | calls | self s/call | total s/call | name |

|–||–|-|–||-| | xxxxx | xxxxx | 15.47 | 3 | 5.16 | 5.16 | colorea | | xxxxx | xxxxx | 1.89 | 5 | 0.38 | 0.38 | interpola| | xxxxx | xxxxx | 1.76 | 1 | 1.76 | 3.65 | traza | | xxxxx | xxxxx | 0.46 | - | - | - | main |

Call graph:

index	% time	self	children	called	name
[1]	100.0	0.46	19.12	-	main [1]
				3/3	colorea
				1/1	traza
[2]	79.0	15.47	0.00	3	colorea [2]
[3]	18.6	1.76	1.89	1	traza [3]
				5/5	interpola
[4]	9.7	1.89	0.00	5	interpola [4]

1. ¿En cuánto tiempo se ejecuta el programa de dibujo?

El tiempo total de ejecución se puede obtener de la suma de los tiempos propios de cada procedimiento:

$$15,47\,s + 1,89\,s + 1,76\,s + 0,46\,s = 19,58\,\text{segundos}$$

2. ¿Cuánto tiempo tarda en ejecutarse el código propio de main()?

El tiempo propio de ejecución de `main()` es directamente el indicado como `self seconds`:

$$0,46\,\text{segundos}$$

3. Relación de llamadas entre los procedimientos y número de ejecuciones

- `main()` llama 3 veces a `colorea()` y 1 vez a `traza()`.
- `traza()` llama 5 veces a `interpola()`.

4. Calcule el nuevo tiempo de ejecución del programa si se elimina el código propio de main() y se reduce a la mitad el tiempo de ejecución del código propio de traza().

El tiempo de ejecución propio de `traza()` se reduce a la mitad:

$$\frac{1,76}{2} = 0,88\,\text{segundos}$$

El nuevo tiempo total sería:

$$15,47\,s + 1,89\,s + 0,88\,s + 0 = 18,24\,\text{segundos}$$

5. ¿Se puede reducir el tiempo de ejecución del programa a 10 segundos sin tocar el procedimiento `colorea()`?

No es posible. El tiempo propio de `colorea()` ya es de 15,47 segundos, por lo que cualquier intento de bajar de 10 segundos afectaría necesariamente a este procedimiento, lo cual no está permitido en el enunciado.

1.13. PROBLEMA 3.12

El resultado de la monitorización de una aplicación informática dedicada al análisis de modelos atmosféricos se muestra a continuación (hay información no disponible):

Flat profile: |% time | cumulative seconds | self seconds | calls | self s/call | total s/call | name | |——|————————|————|——-|————-|————|———-| | 74.4 | 30.16 | 30.16 | 52 | 0.58 | 0.58 | nimbo | | 12.7 | 35.29 | 5.13 | 2 | 2.56 | 2.56 | borrasca | | 8.7 | 38.80 | 3.51 | 2 | 1.75 | 1.75 | lluvia | | 4.3 | 40.56 | 1.76 | 1 | 1.76 | 34.17 | nube |

1. ¿En cuánto tiempo se ejecuta el programa? El tiempo de ejecución total es la suma de los tiempos propios:

$$30,16\,s + 5,13\,s + 3,51\,s + 1,76\,s = 40,56\,\text{segundos}$$

2. ¿Porcentaje del tiempo que consume el procedimiento lluvia()? Calculamos el porcentaje que representa `lluvia` sobre el tiempo total:

$$\frac{3,51}{40,56} \times 100 \approx 8,7\,\%$$

3. ¿Cuál es el procedimiento más lento del programa (incluyendo código propio y subrutinas)? El procedimiento más lento es `nube`, con un tiempo total por llamada (`total s/call`) de 34,17 s.

4. ¿Cuánto tiempo tarda en ejecutarse el código propio de borrasca()? El tiempo propio de `borrasca` es:

$$2,56\,\text{segundos}$$

5. Calcule el nuevo tiempo si el procedimiento nimbo() se mejora 3 veces. El nuevo tiempo propio de `nimbo` sería:

$$\frac{30,16}{3} = 10,05\,\text{segundos}$$

El nuevo tiempo total sería:

$$10,05\,s + 5,13\,s + 3,51\,s + 1,76\,s = 20,45\,\text{segundos}$$

6. Propuesta para reducir el tiempo de ejecución a 20 segundos sin afectar a otros procedimientos Para reducir el tiempo a 20 s, podemos mejorar aún más `nimbo`. La contribución actual de `nimbo` es de 30,16 s y necesitamos reducirla a:

$$20\,s - (5,13 + 3,51 + 1,76)\,s = 9,6\,s$$

La mejora necesaria para `nimbo` es:

$$\frac{30,16}{9,6} \approx 3,14\,\text{veces}$$

Por tanto, mejorando unas 3,2 veces el procedimiento `nimbo` se puede reducir el tiempo de ejecución a 20 segundos.

1.14. PROBLEMA 3.13

Después de conectarse a un sistema informático, un usuario ejecuta las siguientes órdenes con el resultado que se muestra:

```
% uptime
9:50am up 173 days, 23:02, 1 user, load average: 0.00, 0.00, 0.00

% time simulador
real 8m0.70s
user 3m5.20s
sys 0m4.01s
```

1. **¿En qué condición de carga se encuentra el computador (baja, media o alta) en el momento de conexión del usuario?** El computador está en una situación de **baja carga**. Esto se puede concluir observando los valores de `load average`, todos iguales a 0.00, lo que indica que no hay procesos esperando para ejecutarse ni en cola en ese momento.
2. **¿Cuál es el tiempo (en segundos) de ejecución del programa simulador?** El tiempo total de ejecución (`real`) del programa `simulador` es:

$$8\,\text{min} \times 60 + 0,70\,\text{segundos} = 480,7\,\text{segundos}$$

El tiempo efectivo de ejecución de CPU (`user + sys`) es:

$$(3\,\text{min} \times 60 + 5,20) + (0\,\text{min} \times 60 + 4,01) = 185,20 + 4,01 = 189,21\,\text{segundos}$$

3. **¿Encuentra alguna incoherencia en los resultados anteriores? Justifique la respuesta con argumentos sólidos.** Sí, hay una incoherencia clara. A pesar de que la carga

del sistema (`load average`) es nula, el tiempo total (`real`) de ejecución del simulador es de 480,7 segundos, mientras que el tiempo efectivo de CPU (`user + sys`) es de solo 189,21 segundos. Esto implica que el programa estuvo esperando durante:

$$480,7\,\text{segundos} - 189,21\,\text{segundos} = 291,49\,\text{segundos}$$

Esta espera no se justifica en un sistema con carga baja (sin procesos esperando). Podría deberse a otras causas, como el uso intensivo de operaciones de E/S o un error de medición, pero sí existe una contradicción evidente entre la baja carga reportada y la larga espera experimentada.

1.15. PROBLEMA 3.14

En un servidor con S.O. Linux se tiene instalado un monitor de actividad `sar` (system activity reporter). Se sabe que cada activación del monitor implica la ejecución de un total de 450 instrucciones máquina y almacena un total de 1024 bytes de información en el fichero `/var/log/sysstat/saDD` del día DD correspondiente. Si el procesador del equipo tiene una velocidad media de ejecución de 75 MIPS (millions of instructions per second):

a) **¿Qué valor debe tener el periodo de muestreo (en milisegundos) si se quiere una sobrecarga (overhead) del 5 %?**

Primero, calculamos el tiempo de CPU que tarda el monitor en ejecutarse por activación. El tiempo de ejecución de las instrucciones es:

$$\text{Tiempo de CPU (s)} = \frac{\text{Número de instrucciones}}{\text{MIPS} \times 10^6} = \frac{450}{75 \times 10^6} = 6 \times 10^{-6}\,\text{s} = 0,006\,\text{ms}$$

La sobrecarga (overhead) es el porcentaje del tiempo que tarda en ejecutarse respecto al periodo de muestreo:

$$\text{Sobrecarga} = \frac{\text{Tiempo de CPU}}{\text{Periodo}} \times 100$$

Despejando el periodo:

$$\text{Periodo} = \frac{\text{Tiempo de CPU} \times 100}{\text{Sobrecarga}} = \frac{0,006\,\text{ms} \times 100}{5} = 0,12\,\text{ms}$$

b) **Suponiendo ahora que el monitor se activa una vez cada 10 minutos, ¿cuál será el tamaño máximo de cada fichero del directorio `/var/log/sysstat`?**

Número de activaciones al día:

$$\frac{24\,\text{h} \times 60\,\text{min}}{10\,\text{min}} = 144\,\text{activaciones}$$

Tamaño máximo del fichero:

$$144\,\text{activaciones} \times 1024\,\text{bytes} = 147456\,\text{bytes}$$

1.16. PROBLEMA 3.15

Después de instrumentar un programa con la herramienta `gprof`, el resultado obtenido ha sido el siguiente:

Flat profile:

Each sample counts as 0.01 seconds. | % time | cumulative seconds | self seconds | calls | ms/call | ms/call | name | |——|——————|————|——-|———|———|——| | 60.28 | 70.32 | 70.32 | 10 | 7032.00 | 7032.00 | ordena | | 28.96 | 104.10 | 33.78 | 8 | 4222.50 | 4222.50 | inicio | | 10.76 | 116.66 | 12.56 | 34 | 369.41 | 369.41 | escala |

a) **¿Cuántas muestras se han tomado para obtener las estimaciones del tiempo de CPU de cada función?** Cada muestra cuenta como 0.01 segundos. El tiempo total de CPU (`cumulative seconds`) es de 116,66 segundos. Elegimos este valor debido a que es el correspondiente a la medición final, mientras que los anteriores son mediciones intermedias. Por tanto, el número total de muestras es:

$$\frac{116,66}{0,01} = 11666\,\text{muestras}$$

b) **Si el procedimiento más rápido de los tres se sustituye por otro tres veces más rápido, ¿cuánto tiempo tardará en ejecutarse el programa?** El procedimiento más rápido es `escala`, con 12,56 segundos. Mejorándolo 3 veces, su nuevo tiempo sería:

$$\frac{12,56}{3} = 4,19\,\text{segundos}$$

El nuevo tiempo total sería la suma del tiempo mejorado de `escala` y los tiempos de `ordena` e `inicio` que permanecen igual:

$$70,32\,s + 33,78\,s + 4,19\,s = 108,29\,\text{segundos}$$

c) **Si los tres procedimientos se sustituyen por nuevas versiones 2, 4 y 5 veces más rápidas, respectivamente, ¿cuál sería la ganancia en velocidad conseguida con respecto al programa original?** Calculamos los nuevos tiempos:

- `ordena`: $\frac{70,32}{2} = 35,16\,\text{segundos}$
- `inicio`: $\frac{33,78}{4} = 8,445\,\text{segundos}$
- `escala`: $\frac{12,56}{5} = 2,512\,\text{segundos}$

Nuevo tiempo total:

$$35,16\,s + 8,445\,s + 2,512\,s = 46,12\,\text{segundos}$$

La ganancia en velocidad (speedup) es:

$$\frac{116,66}{46,12} \approx 2,53$$

1.17. PROBLEMA 3.16

El resultado de la monitorización de la actividad (usando `gprof` bajo Linux) de una aplicación informática que está siendo ejecutada dentro de un servidor dedicado a streaming de vídeo se muestra a continuación (con información no disponible). El perfil de llamadas indica que todos los procedimientos son llamados únicamente desde el programa principal `main` (que solo se ejecuta una vez y cuyo tiempo propio de ejecución se puede despreciar), excepto `ordena`, que es llamado únicamente desde `procesa`.

Flat profile:

% time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	name
xx	xx	1,8	8	225	225	ordena
xx	xx	1,8	2	900	xx	procesa
xx	xx	1,4	4	350	350	invierte
xx	xx	0,7	4	175	xx	almacena

1.17.1. a) Complete la información no disponible y calcule el tiempo total de CPU

Primero, los tiempos propios (`self seconds`) se suman para obtener el tiempo total de CPU:

$$1,8\,s(\text{ordena}) + 1,8\,s(\text{procesa}) + 1,4\,s(\text{invierte}) + 0,7\,s(\text{almacena}) = 5,7\,\text{segundos}$$

Para completar los porcentajes del tiempo total:

- `ordena`: $\frac{1,8}{5,7} \times 100 \approx 31,6\,\%$
- `procesa`: $\frac{1,8}{5,7} \times 100 \approx 31,6\,\%$
- `invierte`: $\frac{1,4}{5,7} \times 100 \approx 24,6\,\%$
- `almacena`: $\frac{0,7}{5,7} \times 100 \approx 12,3\,\%$

Completamos la tabla con los datos calculados y los tiempos acumulados:

% time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	name
31,6	1,8	1,8	8	225	225	ordena
31,6	3,6	1,8	2	900	1800	procesa
24,6	5,0	1,4	4	350	350	invierte
12,3	5,7	0,7	4	175	175	almacena

La aplicación consume un total de **5,7 segundos de tiempo de CPU**.

1.17.2. b) Calcular la ganancia en velocidad (speedup) al mejorar `ordena` 3 veces

El tiempo propio de `ordena` se reduce a:

$$\frac{1,8}{3} = 0,6\,\text{segundos}$$

El nuevo tiempo total de la aplicación es:

$$0,6\,s(\text{ordena mejorada}) + 1,8\,s(\text{procesa}) + 1,4\,s(\text{invierte}) + 0,7\,s(\text{almacena}) = 4,5\,\text{segundos}$$

El **speedup** es:

$$S = \frac{5,7}{4,5} \approx 1,27$$

Esto corresponde a una mejora del 27 % en la velocidad de ejecución.

Respuestas finales:

- La aplicación consume 5,7 segundos de tiempo de CPU.
- El speedup conseguido mejorando `ordena` 3 veces es 1,27, lo que equivale a un 27 % más de velocidad.

1.18. EJERCICIO 3.11

La monitorización de un programa de dibujo en tres dimensiones mediante la herramienta `gprof` ha proporcionado la siguiente información (por errores en la transmisión hay valores que no están disponibles):

```
Flat profile:
% time cumulative seconds self seconds calls self s/call total s/call name
```

```
xxxxx xxxxx 15.47 3 5.16 5.16 colorea
xxxxx xxxxx 1.89 5 0.38 0.38 interpola
xxxxx xxxxx 1.76 1 1.76 3.65 traza
xxxxx xxxxx 0.46 main

Call graph:
index % time self children called name
[1] 100.0 0.46 19.12 main [1]
    15.47 0.00 3/3 colorea [2]
    1.76 1.89 1/1 traza [3]
    15.47 0.00 3/3 main [1]
[2] 79.0 15.47 0.00 3 colorea [2]
    1.76 1.89 1/1 main [1]
[3] 18.6 1.76 1.89 1 traza [3]
    1.89 0.00 5/5 interpola [4]
    1.89 0.00 5/5 traza [3]
[4] 9.7 1.89 0.00 5 interpola [4]
```

1.18.1. 1. ¿En cuánto tiempo se ejecuta el programa de dibujo?

Hay varias formas de verlo y todas coinciden.

En el **flat profile**, podemos obtener el tiempo total de ejecución como la suma de los tiempos propios (`self seconds`):

$$15,47 + 1,89 + 1,76 + 0,46 = 19,58\,\text{segundos}$$

Desde el **call graph**, como el método `main` se ejecuta el 100 % del tiempo, podemos calcularlo como la suma de su tiempo propio más el de sus hijos:

$$0,46 + 19,12 = 19,58\,\text{segundos}$$

1.18.2. 2. ¿Cuánto tiempo tarda en ejecutarse el código propio de `main()`?

El código propio de `main()` se ejecuta en **0,46 segundos**, que es el valor del campo `self seconds` de la línea correspondiente al procedimiento `main` en el **flat profile**.

1.18.3. 3. Relación de llamadas entre los procedimientos y número de ejecuciones

El procedimiento `main()` llama:

- 3 veces al procedimiento `colorea()`
- 1 vez al procedimiento `traza()`

A su vez, el procedimiento `traza()` llama 5 veces al procedimiento `interpola()`.

1.18.4. 4. Calcule el nuevo tiempo de ejecución si se elimina el código propio de `main()` y se reduce a la mitad el tiempo propio de `traza()`

Reducimos el tiempo propio de `traza()` a la mitad:

$$\frac{1,76}{2} = 0,88 \,\text{segundos}$$

El nuevo tiempo de ejecución sería:

$$15,47 + 1,89 + 0,88 + 0 = 18,24 \,\text{segundos}$$

1.18.5. 5. ¿Se puede conseguir que el programa se ejecute en 10 segundos sin afectar a `colorea()`?

Esto no es posible, puesto que el procedimiento `colorea()` ya consume más de 10 segundos de tiempo de ejecución propio. No se puede reducir su tiempo de ejecución sin afectar a su código o al número de veces que se ejecuta. Por tanto, cualquier acción que se proponga para reducir el tiempo de ejecución total **afectará necesariamente** al procedimiento `colorea()`.