Àlex Aguilera
Miguel Fernández

Práctica medidas de rendimiento

1. Escribir un programa que calcule el tiempo que tarda una llamada a sistema sencilla. Haga una tabla con los tiempos de ejecución que obtenga. ¿Por qué los tiempos de ejecución son tan diferentes?

El código que hemos usado para completar la tabla siguiente, se encuentra en el fichero syscalls.c. Hemos ejecutado las primeras syscalls N=1000000 veces y la syscall fork() N2= 10000 veces, tal y como dice el enunciado.

Syscall	sbrk(0)	sbrk(inc)	sched_yield()	getpid()	fork/waitpid
Execution time (in microsecs.)	3092	226862	302723	190193	1331454

El tiempo de ejecución de cada syscall es diferente y puede variar bastante entre unas y otras. Esto, es debido a que a que algunas llamadas como podrían ser sbrk() o getpid(), son bastante simples y simplemente han de retornar un valor y no hacer ninguna operación de cálculo que pueda incrementar su tiempo de ejecución. Sin embargo, hay otras como ejemplo la llamada a sistema fork(), que debe crear todas las estructuras necesarias para crear un nuevo proceso e inicializarlas debidamente, es por ello que su tiempo de ejecución es considerablemente mayor.

2. Puede comprobar de alguna manera que los programas ejecutan realmente la llamada a sistema? (Y no aprovechan el resultado devuelto por la llamada anterior)

Para comprobar que llamadas a sistema utiliza un programas y ver si realmente se aprovechan del resultado devuelto por otra llamada, podemos usar strace (comentado en el forum del Racó). Para ello, simplemente hay que ejecutar en la consola lo siguiente: strace ./syscalls 0/1/2/3/4 e ir viendo si ejecuta esa syscall o no.

En caso de sbrk(), utilitza la syscall brk. Para sched_yied() y getpid(), utiliza esta misma. En el caso de fork(), utiliza clone() con los parámetros vistos en clase de teoría y para el caso de waitpid(), utilitza wait4(-1,NULL,0,NULL).

3. En la última transparencia del tema Virtualización-Sincronización-MesuresdeRendiment (transp. 50) le pedimos que haga un programa que escriba en el disco 500MBytes, midiendo el tiempo que tarda en hacerlo. Desde un usuario no privilegiado (no root), ejecute su programa sobre un archivo en el disco de la máquina que se utiliza para este laboratorio.

Con los códigos individuales de cada estudiante, hemos obtenido los siguientes diagramas:

	Time	Seconds
	Bandwith	MB

write-to-disk2.py

		Desktop VM 2GB		Desktop 8GB IDE		Laptop VM 8GB
	Time	BandWidth	Time	BandWidth	Time	BandWidth
1	29,1307	17,164	42,8178	11,6773	37,5609	13,3116
2	23,3403	21,4221	31,5712	15,8372	21,6308	23,1151
3	25,241	19,809	32,597	15,3388	22,5475	22,1753
4	22,1873	22,5353	31,247	16,0015	22,4107	22,3106
5	17,2078	29,0565	32,138	15,5579	22,384	22,3373

write-to-disk1.c

		Desktop VM 2GB		Desktop 8GB IDE		Laptop VM 8GB
	Time	BandWidth	Time	BandWidth	Time	BandWidth
1	0,0006	727802,0378	0,0002	2202643,172	0,0001	2688172,043
2	0,0007	635324,0152	0,0003	1497005,988	0,0004	1094091,904
3	0,0013	370370,3703	0,0003	1436781,609	0,0004	1072961,373
4	0,0008	584112,1495	0,0003	1597444,089	0,0004	1133786,848
5	0,0007	628140,7035	0,0003	1661129,568	0,0004	1157407,407

Figura 1. Resultados

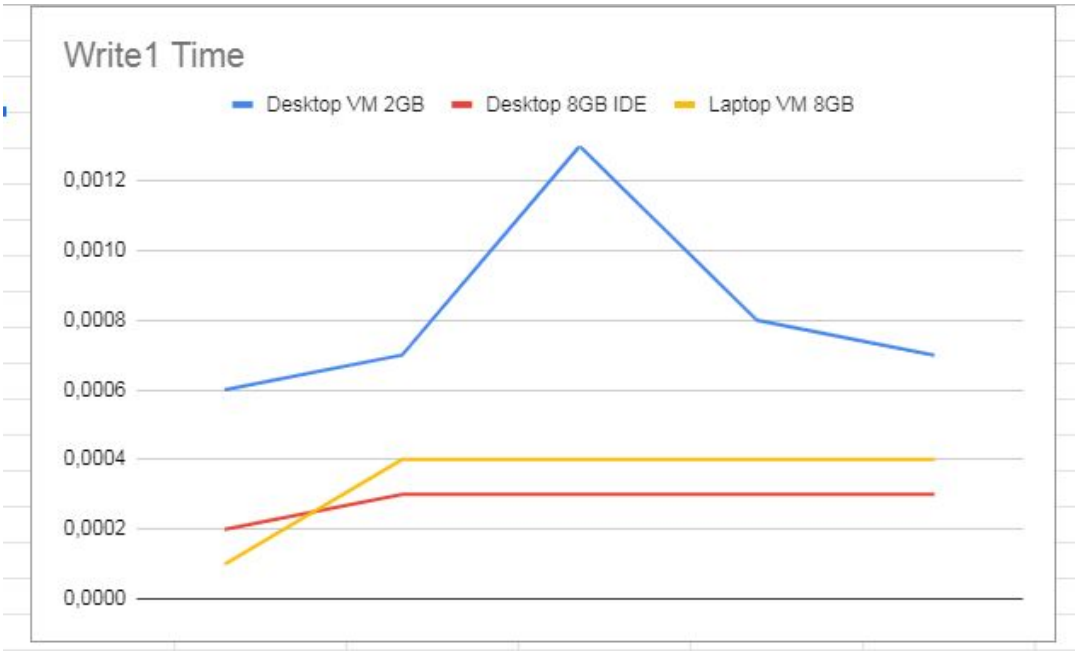

Figura 2. Gráfica Tiempo write-to-disk1.c (segundos)

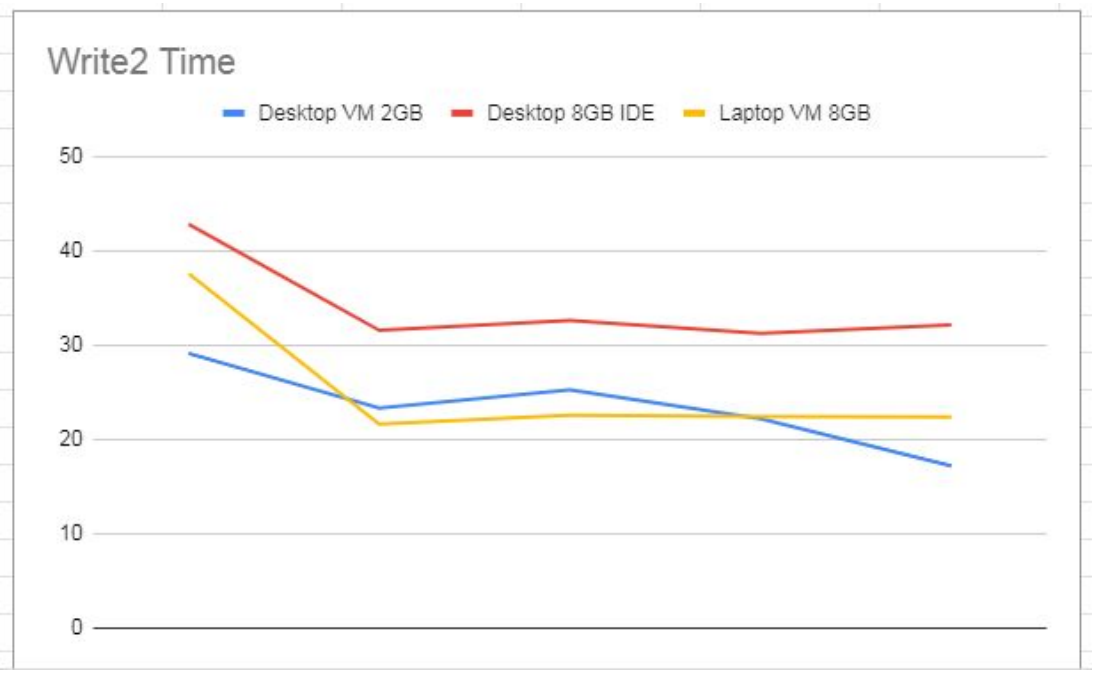

Figura 3. Gráfica Tiempo write-to-disk2.py (segundos)