



PERFORMANCE EN SISTEMAS DE SOFTWARE
2C 2022

TRABAJO PRACTICO N° 1

PADRÓN	ALUMNO	DIRECCIÓN DE CORREO
102896	MOVIA, GUIDO	GMOVIA@FI.UBA.AR

Índice

1. Introducción	2
2. Problemas	2
2.1. CPU caliente	2
2.1.1. Introducción	2
2.1.2. Desarrollo	2
2.1.2.1. Código	2
2.1.2.2. Herramientas	3
2.1.2.3. Análisis	6
2.1.3. Conclusión	9
2.2. Descansando	9
2.2.1. Introducción	9
2.2.2. Desarrollo	9
2.2.2.1. Código	9
2.2.2.2. Herramientas	10
2.2.2.3. Análisis	12
2.2.3. Conclusión	14
2.3. Escritura constante	14
2.3.1. Introducción	14
2.3.2. Desarrollo	14
2.3.2.1. Código	14
2.3.2.2. Herramientas	16
2.3.2.3. Analisis	18
2.3.3. Conclusión	19
3. Conclusión	20

1. Introducción

El presente trabajo práctico examina los problemas causados por tres programas escritos en el lenguaje C, con el objetivo de analizar como la ejecución de cada uno de ellos impacta en el rendimiento del sistema. Para ello, se utilizarán en conjunto algunas de las herramientas que provee la shell de Unix, para obtener información acerca del estado del sistema, sus procesos o dispositivos.

2. Problemas

2.1. CPU caliente

2.1.1. Introducción

El primer problema presenta un programa cuya funcionalidad es ejecutar un bucle infinito.

2.1.2. Desarrollo

Esta sección se encuentra compuesta por tres subsecciones: Código, Herramientas y Análisis. La primera subsección mostrará el código a analizar, mientras que la segunda mostrará las herramientas que se utilizarán para dicho análisis. Finalmente, en la última subsección, se realiza un análisis del problema dado.

2.1.2.1. Código

El código es simple. La función main realiza un llamado a la función loop, la cual se encarga de realizar un bucle infinito a partir de la instrucción for, y es por ello que, nunca va a devolver el número entero.

```
void loop(){  
    for(;;){}  
}  
  
int main(int argc, char *argv[]){  
    loop();  
    return (0);  
}
```

Figura 1: Código asociado al primer problema, escrito en lenguaje C.

2.1.2.2. Herramientas

Las herramientas que se utilizaron para analizar el problema son: pidstat, mpstat y top. A continuación, se explicará como funciona cada una de ellas, su sintaxis, semántica, y el fundamento por el cual fueron seleccionadas.

pidstat El comando pidstat permite monitorear los procesos individuales que actualmente administra el kernel de Linux. Este comando brindará información detallada del proceso, como por ejemplo, estadísticas de uso de CPU, entrada/salida, fallas de página, uso de memoria, entre otros. La siguiente imagen permite visualizar la sintaxis del comando pidstat, extraída del manual de Linux.

Seleccionamos el comando pidstat, ya que nos permitirá obtener información sobre el uso de la CPU que realiza el proceso ejecutado, el cual, realiza un bucle infinito. Para ello, utilizaremos el flag -p para filtrar por el ID del proceso a analizar, y el comando -u para obtener los reportes de la CPU. Tras la ejecución del comando pidstat -u -p ID, obtendremos una salida compuesta por diversas columnas, cuyo significado explicaremos a continuación:

- UID: Número de identificación del usuario que está ejecutando la tarea.
- USER: Nombre del usuario real que está supervisando la tarea.
- PID: Número de identificación de la tarea que está siendo monitoreada.
- %usr: Porcentaje de CPU utilizado por la tarea mientras se ejecuta en modo usuario.
- %system: Porcentaje de CPU utilizado por la tarea mientras se ejecuta en modo kernel.

- %guest: Porcentaje de CPU utilizado por la tarea en máquina virtual.
- %wait: Porcentaje de CPU utilizado por la tarea mientras espera.
- %CPU: Porcentaje total de tiempo de CPU utilizado por la tarea.
- CPU: Número del procesador al que está asociado la tarea.
- Command: El nombre de la tarea.

A continuacion, se muestra un ejemplo del uso de pidstat para analizar el proceso que ejecuta Bash.

```
gmovia@gmovia-System-Product-Name:~$ pidstat -u -p 7374
Linux 5.15.0-48-generic (gmovia-System-Product-Name) 05/10/22 _x86_64_ (12 CPU)

12:32:51      UID      PID    %usr %system  %guest  %wait   %CPU   CPU  Command
12:32:51    1000    7374    0,00  0,00    0,00   0,00   0,00    8  bash
gmovia@gmovia-System-Product-Name:~$
```

Figura 2: Ejemplo de uso de pidstat.

mpstat El comando mpstat se utiliza para informar estadísticas relacionadas con el procesador. Muestra con precisión las estadísticas del uso de CPU del sistema, la utilización y rendimiento del CPU, entre otras cosas.

Seleccionamos mpstat ya que nos permitirá brindar información sobre el estado actual de la CPU. Es útil brindarle los parámetros TIME y COUNT, ya que nos permitirá obtener un reporte de la CPU cada un período TIME de tiempo, hasta obtener una cantidad COUNT de reportes. Las columnas que nos brindara mpstat son:

- CPU: Número de procesador de la CPU. La palabra all indica que las estadísticas se calculan como un promedio entre todos los procesadores.
- %usr: Porcentaje de uso de CPU mientras se ejecutaba a nivel de usuario.
- %nice: Muestre el porcentaje de utilización de la CPU que se produjo durante la ejecución a nivel de usuario con buena prioridad.
- %sys: Porcentaje de uso de CPU que ocurrió mientras se ejecutaba en modo kernel.
- %iowait: Porcentaje de tiempo que la CPU estuvo inactiva durante los cuales el sistema tuvo solicitudes E/S en disco pendiente.
- %irq: Porcentaje de tiempo empleado por la CPU para dar servicio a las interrupciones de hardware.
- %soft: Porcentaje de tiempo empleado por la CPU para dar servicio a las interrupciones por software.

- %guest: Porcentaje de tiempo empleado por la CPU para ejecutar un procesador virtual.
- %gnice: Muestra el porcentaje de tiempo empleado por la CPU o las CPU para ejecutar un invitado agradable.
- %idle: Porcentaje de tiempo que la CPU estaba inactiva y el sistema no tenia solicitud de E/S en disco pendiente.

A continuación, se muestra un ejemplo del uso de mpstat, el cual realiza un reporte cada cuatro segundos hasta informar tres.

```
gmovia@gmovia-System-Product-Name:~$ mpstat 4 3
Linux 5.15.0-48-generic (gmovia-System-Product-Name) 05/10/22 _x86_64_ (12 CPU)

12:57:42 CPU %usr %nice %sys %iowait %irq %soft %steal %guest %gnice %idle
12:57:46 all 1,63 0,00 0,36 0,17 0,00 0,06 0,00 0,00 0,00 97,78
12:57:50 all 4,44 0,00 0,98 0,04 0,00 0,23 0,00 0,00 0,00 94,30
12:57:54 all 3,36 0,00 0,74 0,06 0,00 0,04 0,00 0,00 0,00 95,80
Media: all 3,14 0,00 0,69 0,09 0,00 0,11 0,00 0,00 0,00 95,96
gmovia@gmovia-System-Product-Name:~$
```

Figura 3: Ejemplo de uso de mpstat.

top El comando top permite ver las tareas del sistema que se ejecutan en tiempo real. Produce una lista ordenada de procesos en ejecución, seleccionados por criterios especificados por el usuario. El orden predeterminado es por uso de CPU, y solo se muestran los principales consumidores de CPU. Top es útil para administradores del sistema, ya que muestra qué usuarios y qué procesos están consumiendo la mayor cantidad de recursos del sistema en un momento dado. Es por ello que lo utilizaremos.

Las columnas que nos brinda el comando top son las siguientes:

- PID: Identificador del proceso.
- PR: La prioridad de programación de la tarea.
- NI: El valor de prioridad de la tarea. Si es negativo, significa que posee mayor prioridad. Si es positivo, es porque significa que posee menor prioridad.
- VIRT: Cantidad de memoria virtual utilizada por la tarea, en KiB.
- RES: Cantidad de memoria RAM física que utiliza el proceso.
- SHR: Tamaño de la memoria compartida con otros procesos.
- S: R (proceso que está corriendo o se encuentra en la cola de ejecución), S (procesos que están esperando un evento para su ejecución), D (procesos que están esperando a que termine una operación I/O), T (proceso detenido), Z (proceso zombie)

- %CPU: Capacidad de procesamiento que consume cada uno de los procesos o tareas.
- %MEM: Porcentaje de memoria RAM que está consumiendo cada uno de los procesos o tareas.
- HORA+: Tiempo total de CPU que ha usado una tarea o proceso desde que se inicio.
- ORDEN: Muestra el nombre del proceso o el comando usado para iniciar la tarea.

A continuación, se muestra un ejemplo del uso del comando top.

```
top - 13:25:28 up 2:55, 1 user, load average: 0,59, 0,98, 1,04
Tareas: 353 total, 1 ejecutar, 352 hibernar, 0 detener, 0 zombie
%Cpu(s): 1,1 us, 0,3 sy, 0,0 ni, 98,5 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
MiB Mem : 7874,2 total, 1269,6 libre, 3129,6 usado, 3475,1 búfer/caché
MiB Intercambio: 2048,0 total, 2048,0 libre, 0,0 usado, 4252,2 dispon Mem
```

PID	USUARIO	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	HORA+	ORDEN
7108	gmovia	20	0	1130,0g	293824	131216	S	7,0	3,6	5:36.07	chrome
1457	gmovia	20	0	6216068	338660	148780	S	3,6	4,2	5:49.71	gnome-shell
4736	gmovia	20	0	32,6g	363544	194540	S	3,3	4,5	4:23.66	chrome
4863	gmovia	20	0	32,8g	309736	178476	S	3,3	3,8	7:40.31	chrome
11377	gmovia	20	0	1129,5g	399872	130140	S	2,3	5,0	5:04.93	chrome
1271	gmovia	9	-11	2287036	29672	22920	S	1,7	0,4	0:53.21	pulseaudio
2530	gmovia	20	0	2548996	106700	73580	S	1,0	1,3	0:11.98	xdg-desktop-por
4769	gmovia	20	0	1414732	110268	65252	S	1,0	1,4	1:06.79	Xwayland
1377	root	20	0	0	0	0	S	0,7	0,0	0:19.95	RTW_CMD_THREAD
1573	mongodb	20	0	2614616	125520	61700	S	0,7	1,6	0:55.21	mongod
5800	gmovia	20	0	32,8g	75528	63376	S	0,7	0,9	0:20.39	chrome

Figura 4: Ejemplo de uso de top.

2.1.2.3. Análisis

Empecemos por ver el estado de la CPU antes de la ejecución del código. Para ello, utilizamos el comando top. El resultado de la ejecución del comando nos brinda la siguiente salida.

PID	USUARIO	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	HORA+	ORDEN
7108	gmovia	20	0	1130,0g	294120	132808	S	15,9	3,6	7:31.14	chrome
4863	gmovia	20	0	32,8g	321420	185128	S	4,3	4,0	9:35.66	chrome
1457	gmovia	20	0	6228920	354588	148352	S	2,7	4,4	6:39.99	gnome-shell
11377	gmovia	20	0	1129,7g	450424	129232	S	2,7	5,6	9:38.75	chrome
4736	gmovia	20	0	32,6g	377768	196396	S	2,0	4,7	5:16.85	chrome
1271	gmovia	9	-11	2287036	29296	22528	S	1,0	0,4	1:13.40	pulseaudio
969	root	20	0	2056820	42864	19120	S	0,7	0,5	0:09.66	snappd
4769	gmovia	20	0	1420672	110272	65252	S	0,7	1,4	1:20.36	Xwayland
4866	gmovia	20	0	32,4g	125116	92836	S	0,7	1,6	1:10.55	chrome
5800	gmovia	20	0	32,8g	78296	66224	S	0,7	1,0	0:28.27	chrome
1573	mongodb	20	0	2614616	125520	61700	S	0,3	1,6	1:10.32	mongod
14364	gmovia	20	0	16080	4212	3360	R	0,3	0,1	0:13.24	top
1	root	20	0	168084	13536	8352	S	0,0	0,2	0:01.99	systemd
2	root	20	0	0	0	0	S	0,0	0,0	0:00.02	kthreadd
3	root	0	-20	0	0	0	I	0,0	0,0	0:00.00	rcu_gp
4	root	0	-20	0	0	0	I	0,0	0,0	0:00.00	rcu_par_gp
5	root	0	-20	0	0	0	I	0,0	0,0	0:00.00	netns
7	root	0	-20	0	0	0	I	0,0	0,0	0:00.00	kworker/0:0H-events_highpri
9	root	0	-20	0	0	0	I	0,0	0,0	0:00.03	kworker/0:1H-kblockd
10	root	0	-20	0	0	0	I	0,0	0,0	0:00.00	mm_percpu_wq

Figura 5: Salida de top antes de la ejecución del programa.

Ahora bien, ¿Que sucederá cuando ejecutamos el código? A priori, podemos pensar que como el código realiza un bucle infinito, entonces consumirá todo el tiempo del procesador. Si ejecutamos el código entonces el comando top se actualiza, brindando la siguiente salida

PID	USUARIO	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	HORA+	ORDEN
17276	gmovia	20	0	2640	940	852	R	100,0	0,0	0:17.95	a.out
1271	gmovia	9	-11	2287036	29548	22780	S	2,0	0,4	1:14.78	pulseaudio
11377	gmovia	20	0	1129,7g	446040	129232	S	2,0	5,5	9:41.32	chrome
1377	root	20	0	0	0	0	D	0,7	0,0	0:28.07	RTW_CMD_THREAD
1457	gmovia	20	0	6232484	354708	148372	S	0,7	4,4	6:43.48	gnome-shell
5800	gmovia	20	0	32,8g	78296	66224	S	0,7	1,0	0:28.83	chrome
1	root	20	0	168084	13536	8352	S	0,3	0,2	0:02.00	systemd
1573	mongodb	20	0	2614616	125520	61700	S	0,3	1,6	1:10.72	mongod
2165	gmovia	20	0	391536	12152	7436	S	0,3	0,2	0:15.99	ibus-daemon
2460	gmovia	20	0	166392	7644	7044	S	0,3	0,1	0:05.11	ibus-engine-sim
4736	gmovia	20	0	32,6g	377524	196048	S	0,3	4,7	5:18.49	chrome
4900	gmovia	20	0	32,3g	50332	36716	S	0,3	0,6	0:02.51	chrome
7356	gmovia	20	0	566748	63280	44384	S	0,3	0,8	0:23.75	gnome-terminal-
14364	gmovia	20	0	16080	4212	3360	R	0,3	0,1	0:13.57	top
15933	root	20	0	0	0	0	I	0,3	0,0	0:00.32	kworker/u64:0-events_power_e
2	root	20	0	0	0	0	S	0,0	0,0	0:00.02	kthreadd
3	root	0	-20	0	0	0	I	0,0	0,0	0:00.00	rcu_gp
4	root	0	-20	0	0	0	I	0,0	0,0	0:00.00	rcu_par_gp
5	root	0	-20	0	0	0	I	0,0	0,0	0:00.00	netns
7	root	0	-20	0	0	0	I	0,0	0,0	0:00.00	kworker/0:0H-events_highpri

Figura 6: Salida de top durante la ejecución del programa.

Podemos notar que efectivamente, el programa consumirá todo el tiempo del procesador, validando lo que se pensaba en un primer punto. Podemos notar revisando las columnas restantes que: el proceso está en ejecución, su id es 17276, no está consumiendo memoria RAM, etc.

Veamos también que información nos puede agregar el comando mpstat. Para ello, repetimos el procedimiento que realizamos anteriormente.

```
gmovia@gmovia-System-Product-Name:~/Escritorio/PerformanceTP$ mpstat 2 15
Linux 5.15.0-48-generic (gmovia-System-Product-Name) 05/10/22 _x86_64_(12 CPU)
```

	CPU	%usr	%nice	%sys	%iowait	%irq	%soft	%steal	%guest	%gnice	%idle
14:32:33	all	0,46	0,00	0,17	0,21	0,00	0,04	0,00	0,00	0,00	99,12
14:32:35	all	0,84	0,00	0,50	0,00	0,00	0,00	0,00	0,00	0,00	98,66
14:32:37	all	0,46	0,00	0,25	0,46	0,00	0,00	0,00	0,00	0,00	98,83
14:32:39	all	0,54	0,00	0,21	0,38	0,00	0,00	0,00	0,00	0,00	98,87
14:32:41	all	1,92	0,00	0,58	0,00	0,00	0,08	0,00	0,00	0,00	97,42
14:32:43	all	1,46	0,00	0,63	0,83	0,00	0,04	0,00	0,00	0,00	97,04
14:32:45	all	1,26	0,00	0,50	0,00	0,00	0,04	0,00	0,00	0,00	98,20
14:32:47	all	1,51	0,00	0,34	0,08	0,00	0,00	0,00	0,00	0,00	98,07
14:32:49	all	0,92	0,00	0,29	0,00	0,00	0,04	0,00	0,00	0,00	98,75
14:32:51	all	0,42	0,00	0,21	0,25	0,00	0,00	0,00	0,00	0,00	99,12
14:32:53	all	0,33	0,00	0,17	0,08	0,00	0,00	0,00	0,00	0,00	99,42
14:32:55	all	0,54	0,00	0,33	0,00	0,00	0,00	0,00	0,00	0,00	99,13
14:32:57	all	0,46	0,00	0,13	0,00	0,00	0,04	0,00	0,00	0,00	99,37
14:32:59	all	0,29	0,00	0,25	0,33	0,00	0,04	0,00	0,00	0,00	99,08
14:33:01	all	0,83	0,00	0,21	0,08	0,00	0,00	0,00	0,00	0,00	98,87
14:33:03	Media:	0,82	0,00	0,32	0,18	0,00	0,02	0,00	0,00	0,00	98,66

```
gmovia@gmovia-System-Product-Name:~/Escritorio/PerformanceTP$
```

Figura 7: Salida de mpstat antes de la ejecución del programa.


```
gmovia@gmovia-System-Product-Name:~/Escritorio/PerformanceTP$ mpstat 2 15
Linux 5.15.0-48-generic (gmovia-System-Product-Name) 05/10/22 _x86_64_ (12 CPU)
```

	CPU	%usr	%nice	%sys	%iowait	%irq	%soft	%steal	%guest	%gnice	%idle
14:30:27	all	8,89	0,00	0,13	0,63	0,00	0,04	0,00	0,00	0,00	90,32
14:30:29	all	8,89	0,00	0,13	0,63	0,00	0,04	0,00	0,00	0,00	90,32
14:30:31	all	8,50	0,00	0,29	0,33	0,00	0,13	0,00	0,00	0,00	90,75
14:30:33	all	8,59	0,00	0,17	0,00	0,00	0,08	0,00	0,00	0,00	91,16
14:30:35	all	8,45	0,00	0,08	0,25	0,00	0,00	0,00	0,00	0,00	91,21
14:30:37	all	10,27	0,00	0,46	0,42	0,00	0,00	0,00	0,00	0,00	88,86
14:30:39	all	10,69	0,00	0,50	0,70	0,00	0,33	0,00	0,00	0,00	87,77
14:30:41	all	8,84	0,00	0,25	0,04	0,00	0,17	0,00	0,00	0,00	90,70
14:30:43	all	8,85	0,00	0,17	0,33	0,00	0,25	0,00	0,00	0,00	90,40
14:30:45	all	8,44	0,00	0,08	0,17	0,00	0,04	0,00	0,00	0,00	91,27
14:30:47	all	8,47	0,00	0,17	0,00	0,00	0,08	0,00	0,00	0,00	91,28
14:30:49	all	8,72	0,00	0,17	0,00	0,00	0,00	0,00	0,00	0,00	91,12
14:30:51	all	2,89	0,00	0,13	0,08	0,00	0,04	0,00	0,00	0,00	96,86
14:30:53	all	3,13	0,00	0,38	0,00	0,00	0,00	0,00	0,00	0,00	96,50
14:30:55	all	0,29	0,00	0,37	0,00	0,00	0,00	0,00	0,00	0,00	99,33
14:30:57	all	0,33	0,00	0,21	0,08	0,00	0,04	0,00	0,00	0,00	99,33
Media:	all	7,03	0,00	0,24	0,20	0,00	0,08	0,00	0,00	0,00	92,46

```
gmovia@gmovia-System-Product-Name:~/Escritorio/PerformanceTP$
```

Figura 8: Salida de mpstat durante la ejecución del programa.

Podemos notar que las variables que se vieron más afectadas son %usr y %idle. Recordemos que la primera de ellas representa el porcentaje de uso de CPU mientras se ejecutaba a nivel usuario, y la segunda es el porcentaje de tiempo que la CPU estaba inactiva y que el sistema no tenía solicitud E/S en disco pendiente.

Finalmente, utilizamos el comando pidstat para obtener información adicional sobre el proceso que se está ejecutando.

```
gmovia@gmovia-System-Product-Name:~/Escritorio/PerformanceTP$ pidstat -p 18924 2 10
Linux 5.15.0-48-generic (gmovia-System-Product-Name) 05/10/22 _x86_64_ (12 CPU)
```

	UID	PID	%usr	%system	%guest	%wait	%CPU	CPU	Command
14:42:03	1000	18924	100,00	0,00	0,00	0,00	100,00	6	a.out
14:42:05	1000	18924	100,00	0,00	0,00	0,00	100,00	5	a.out
14:42:07	1000	18924	100,00	0,00	0,00	0,00	100,00	11	a.out
14:42:09	1000	18924	99,50	0,00	0,00	0,00	99,50	5	a.out
14:42:11	1000	18924	100,00	0,00	0,00	0,00	100,00	5	a.out
14:42:13	1000	18924	100,00	0,00	0,00	0,00	100,00	6	a.out
14:42:15	1000	18924	100,00	0,00	0,00	0,00	100,00	3	a.out
14:42:17	1000	18924	100,00	0,00	0,00	0,50	100,00	9	a.out
14:42:19	1000	18924	100,00	0,00	0,00	0,00	100,00	3	a.out
14:42:21	1000	18924	100,00	0,00	0,00	0,00	100,00	5	a.out
14:42:23	1000	18924	100,00	0,00	0,00	0,00	100,00	-	a.out
Media:	1000	18924	99,95	0,00	0,00	0,05	99,95	-	a.out

```
gmovia@gmovia-System-Product-Name:~/Escritorio/PerformanceTP$
```

Figura 9: Salida de pidstat durante la ejecución del programa.

Podemos notar que el programa ejecutado tiene el identificador 18924, y se encuentra utilizando todo el porcentaje de la CPU, ejecutandose en todo momento en modo usuario.

2.1.3. Conclusión

El bucle infinito provoca que la aplicación consuma una gran cantidad de recursos del procesador, afectando la performance del sistema. Los comandos seleccionados nos permitieron identificar rápidamente el problema, sin embargo, recomendamos el comando `top` por sobre los demás, ya que nos permite tener una visión rápida y clara, acerca del uso de la CPU en relación a los procesos que se encuentran actualmente activos.

2.2. Descansando

2.2.1. Introducción

El segundo problema presenta un programa cuya funcionalidad es la de realizar un bucle infinito, en el que por cada ciclo, el proceso queda dormido por un periodo de tiempo para luego despertarse, realizar un bucle y finalizar el ciclo.

2.2.2. Desarrollo

Esta sección se encuentra compuesta por tres subsecciones: Código, Herramientas y Análisis. La primera subsección mostrará el código a analizar, mientras que la segunda mostrará las herramientas que se utilizarán para dicho análisis. Finalmente, en la última subsección, se realiza un análisis del problema dado.

2.2.2.1. Código

```
#include <unistd.h>
#define SPINS    (10 * 1000 * 1000) /* tune to ~1% CPU */

void func_l(){
    int i, j;
    for(;;) {
        usleep(1000 * 1000);
        for (i = 0, j = 0; i < SPINS; i++) { j++; }
    }
}

int main(int argc, char *argv[]){
    func_l();
    return (0);
}
```

Figura 10: Código asociado al segundo problema, escrito en lenguaje C.

2.2.2.2. Herramientas

top El comando top permite ver las tareas del sistema que se ejecutan en tiempo real. Produce una lista ordenada de procesos en ejecución, seleccionados por criterios especificados por el usuario. El orden predeterminado es por uso de CPU, y solo se muestran los principales consumidores de CPU. Utilizaremos top para poder conocer el consumo de CPU que utiliza el programa a ejecutar. Para más información acerca de este comando, repasar la sección Herramientas del problema anterior.

pidstat El comando pidstat permite monitorear los procesos individuales que actualmente administra el kernel de Linux. Este comando brindará información detallada del proceso, como por ejemplo, estadísticas de uso de CPU, entrada/salida, fallas de página, uso de memoria, entre otros. La siguiente imagen permite visualizar la sintaxis del comando pidstat, extraída del manual de Linux. Utilizamos pidstat para realizar un seguimiento mas detallado del proceso. Para mas informacion acerca de este comando, repasar la seccion Herramientas del problema anterior.

strace El comando strace permite vigilar las llamadas al sistema usadas por un determinado programa, y todas las señales que éste recibe. Strace se encarga de mostrar el nombre de cada llamada al sistema, junto con sus argumentos y su valor de retorno a error estándar.

Seleccionamos el comando strace ya que nos permitirá obtener información acerca de las llamadas al sistema que realiza el programa. En este caso, nos interesa validar las llamadas a sleep. Utilizaremos los atributos -p para filtrar por el id del proceso y -c para obtener estadísticas. Las estadísticas brindadas son:

- %time:
- seconds:
- usecs/call:
- calls: Cantidad de veces que se llamo a la syscall.
- errors: Cantidad de errores que produjo la syscall.
- syscall: Nombre de la syscall.

A continuacion, se muestra un ejemplo del uso de strace enviando por parametro el ejecutable (tambien se podria utilizar la id con el atributo -p). Es importante mencionar que los reportes se actualizan constantemente hasta la finalizacion del programa, o una

2.2.2.3. Análisis

El código del programa es similar al código asociado al problema de la CPU caliente. En ese caso concluimos que el proceso tomó el control de la CPU, utilizando el 100 % de su tiempo de procesamiento. Ahora bien, ¿Qué sucede en este caso? Para ello, analizaremos la salida del comando top.

PID	USUARIO	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	HORA+	ORDEN
33473	gmovia	20	0	1129,8g	336816	138296	S	21,2	4,2	4:11.05	chrome
34362	gmovia	20	0	2640	988	896	S	4,3	0,0	0:00.27	a.out
8634	gmovia	20	0	1129,5g	375956	127448	S	1,7	4,7	31:25.46	chrome
1271	gmovia	9	-11	2549352	26968	20072	S	1,3	0,3	3:12.62	pulseaudio
4276	gmovia	20	0	32,6g	402852	184784	S	1,0	5,0	8:20.76	chrome
4402	gmovia	20	0	32,8g	342388	189380	S	1,0	4,2	11:44.00	chrome
1571	mongodb	20	0	2609424	121168	58768	S	0,7	1,5	1:18.76	mongod
4486	gmovia	20	0	32,4g	137712	91992	S	0,7	1,7	2:45.39	chrome
5005	gmovia	20	0	32,8g	76960	64828	S	0,7	1,0	1:11.56	chrome
14	root	20	0	0	0	0	I	0,3	0,0	0:11.68	rcu_sched
1474	gmovia	20	0	6135204	342260	136060	S	0,3	4,2	6:31.90	gnome-shell
8096	gmovia	20	0	36,4g	68628	40756	S	0,3	0,9	0:12.77	code
8796	gmovia	20	0	645580	64900	44872	S	0,3	0,8	0:23.61	gnome-terminal-
9779	gmovia	20	0	1129,4g	255652	122852	S	0,3	3,2	1:06.40	chrome
30838	gmovia	20	0	1130,6g	313044	123372	S	0,3	3,9	2:38.04	chrome
34358	gmovia	20	0	16080	4348	3496	R	0,3	0,1	0:00.06	top
1	root	20	0	166744	11256	7616	S	0,0	0,1	0:01.57	systemd
2	root	20	0	0	0	0	S	0,0	0,0	0:00.02	kthreadd
3	root	0	-20	0	0	0	I	0,0	0,0	0:00.00	rcu_gp
4	root	0	-20	0	0	0	I	0,0	0,0	0:00.00	rcu_par_gp
5	root	0	-20	0	0	0	I	0,0	0,0	0:00.00	netns
7	root	0	-20	0	0	0	I	0,0	0,0	0:00.00	kworker/0:0H-events_highpri
9	root	0	-20	0	0	0	I	0,0	0,0	0:00.06	kworker/0:1H-kblockd
10	root	0	-20	0	0	0	I	0,0	0,0	0:00.00	mm_percpu_wq
11	root	20	0	0	0	0	S	0,0	0,0	0:00.00	rcu_tasks_rude_
12	root	20	0	0	0	0	S	0,0	0,0	0:00.00	rcu_tasks_trace
13	root	20	0	0	0	0	S	0,0	0,0	0:00.11	ksoftirqd/0
15	root	rt	0	0	0	0	S	0,0	0,0	0:00.03	migration/0
16	root	-51	0	0	0	0	S	0,0	0,0	0:00.00	idle_inject/0
18	root	20	0	0	0	0	S	0,0	0,0	0:00.00	cpuhp/0
19	root	20	0	0	0	0	S	0,0	0,0	0:00.00	cpuhp/1

Figura 13: Salida de top durante la ejecución del programa.

Podemos notar que en esta oportunidad, el proceso no toma todo el tiempo de ejecución del procesador, sino que solamente el 4,3 %. Veamos que sucede con un seguimiento mas detallado utilizando pidstat.

```
gmovia@gmovia-System-Product-Name:~$ pidstat -p 34362 2 15
Linux 5.15.0-48-generic (gmovia-System-Product-Name) 07/10/22 _x86_64_ (12 CPU)
```

	UID	PID	%usr	%system	%guest	%wait	%CPU	CPU	Command
16:15:07									
16:15:09	1000	34362	5,00	0,00	0,00	0,00	5,00	3	a.out
16:15:11	1000	34362	5,00	0,00	0,00	0,00	5,00	4	a.out
16:15:13	1000	34362	5,00	0,00	0,00	0,00	5,00	4	a.out
16:15:15	1000	34362	4,50	0,00	0,00	0,00	4,50	4	a.out
16:15:17	1000	34362	5,00	0,00	0,00	0,00	5,00	5	a.out
16:15:19	1000	34362	4,50	0,00	0,00	0,00	4,50	5	a.out
16:15:21	1000	34362	3,00	0,00	0,00	0,00	3,00	5	a.out
16:15:23	1000	34362	5,00	0,00	0,00	0,00	5,00	11	a.out
16:15:25	1000	34362	4,50	0,00	0,00	0,00	4,50	11	a.out
16:15:27	1000	34362	5,00	0,00	0,00	0,00	5,00	11	a.out
16:15:29	1000	34362	5,00	0,00	0,00	0,00	5,00	11	a.out
16:15:31	1000	34362	4,50	0,00	0,00	0,00	4,50	11	a.out
16:15:33	1000	34362	5,00	0,00	0,00	0,00	5,00	11	a.out
16:15:35	1000	34362	4,50	0,00	0,00	0,00	4,50	11	a.out
16:15:37	1000	34362	5,00	0,00	0,00	0,00	5,00	11	a.out
Media:	1000	34362	4,70	0,00	0,00	0,00	4,70	-	a.out

```
gmovia@gmovia-System-Product-Name:~$
```

Figura 14: Salida de pidstat durante la ejecución del programa.

Observemos que se valida lo mencionado anteriormente, el proceso utiliza aproximadamente el 5% del tiempo de procesamiento del CPU. Podemos notar que esto sucede ya que el proceso se encuentra descansando durante la mayor parte del tiempo, ya que utiliza la syscall `usleep(useconds_t usec)` que suspende la ejecución de la llamada por usec microsegundos.

Utilizamos el comando `strace` para poder obtener información acerca de las llamadas al sistema realizadas por el proceso. Como el programa realiza un ciclo infinito y no terminará nunca, entonces luego de un periodo de tiempo interrumpimos el comando. La salida se encuentra dada por.

```
gmovia@gmovia-System-Product-Name:~/Escritorio/Proyectos/PerformanceTP$ strace -c ./a.out
^Cstrace: Process 36354 detached
```

% time	seconds	usecs/call	calls	errors	syscall
41,24	0,000353	353	1		execve
23,13	0,000198	0	575	1	clock_nanosleep
11,68	0,000100	12	8		mmap
4,09	0,000035	11	3		mprotect
3,27	0,000028	7	4		pread64
2,92	0,000025	12	2		openat
2,34	0,000020	20	1		munmap
1,99	0,000017	8	2		newfstatat
1,52	0,000013	13	1	1	access
1,40	0,000012	6	2		close
1,40	0,000012	6	2	1	arch_prctl
1,05	0,000009	9	1		read
0,93	0,000008	8	1		prlimit64
0,82	0,000007	7	1		set_tid_address
0,82	0,000007	7	1		set_robust_list
0,70	0,000006	6	1		brk
0,70	0,000006	6	1		rseq
100,00	0,000856	1	607	3	total

```
gmovia@gmovia-System-Product-Name:~/Escritorio/Proyectos/PerformanceTP$
```

Figura 15: Salida de strace durante la ejecución del programa.

Efectivamente, el proceso realiza en casi su totalidad, llamadas a la syscall `clock_nanosleep`, la cual suspende la ejecución del programa en intervalos de microsegundos.

2.2.3. Conclusión

El proceso no consume todos los recursos del procesador que lo esta ejecutando, y esto se debe a que permanece descansando durante la mayor parte del tiempo gracias al llamado de la syscall `usleep`. Tanto el comando `top` como `pidstat` brindaron la informacion correspondiente al consumo del procesamiento, permitiendo analizar la performance del programa. Por otro lado, el comando `strace` nos permitio conocer las llamadas realizadas por el proceso, junto con estadisticas asociadas a las mismas.

2.3. Escritura constante

2.3.1. Introducción

El tercer problema presenta un programa cuya funcionalidad principal es la de realizar una escritura constante en disco. La función `main` se encarga de abrir el archivo, obtener su file descriptor y llamar a la función `write_log()`, la cual realiza un bucle infinito en el que, en cada ciclo, realiza un nuevo loop que se encarga de escribir en disco.

2.3.2. Desarrollo

Esta sección se encuentra compuesta por tres subsecciones: Código, Herramientas y Análisis. La primera subsección mostrará el código a analizar, mientras que la segunda mostrará las herramientas que se utilizarán para dicho análisis. Finalmente, en la última subsección, se realiza un análisis del problema dado.

2.3.2.1. Código

```
/*
 * lab005 - sync disk writes to a file.
 *
 * 21-May-2015  Brendan Gregg  Created this.
 */

#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

const char *datafile = "lab005.data";
#define BUFSIZE      (8 * 1024)
#define FILESIZE     (10 * 1024 * 1024)

void write_log(int fd){
    char buf[BUFSIZE];
    int i;

    for (;;) {
        for (i = 0; i < FILESIZE / BUFSIZE; i++) {
            if (write(fd, buf, BUFSIZE) < 0) {
                printf("ERROR: write error.\n");
                exit(2);
            }
        }

        if (lseek(fd, 0, SEEK_SET) < 0) {
            printf("ERROR: seek() failed.\n");
            exit(3);
        }
    }
}

int main(){
    int fd;

    if ((fd = open(datafile, O_CREAT | O_WRONLY | O_SYNC, 0644)) < 0) {
        printf("ERROR: writing to %s\n", datafile);
        exit(1);
    }

    write_log(fd);

    return (0);
}
```

Figura 16: Código asociado al tercer problema, escrito en lenguaje C.

2.3.2.2. Herramientas

iostat El comando iostat se utiliza para monitorizar la carga del dispositivo de entrada/salida del sistema, observando el tiempo que los dispositivos están activos en relación con sus tasas de transferencia promedio. También, se puede utilizar para comparar la actividad entre discos. El comando iostat genera dos tipos de informes; de utilización de CPU y de utilización del dispositivo.

Las columnas que nos brinda el comando iostat con respecto a los dispositivos de entrada y salida son las siguientes:

- tps: Cantidad de solicitudes de E/S al dispositivo.
- kB_reads/s: Indica la cantidad de datos leídos del dispositivo expresados en kilobytes por segundo.
- kB_wrtn/s: Indica la cantidad de datos escritos en el dispositivo expresados en kilobytes por segundo.
- kB_dscd/s:
- kB_read: Cantidad de kilobytes leídos.
- kB_wrtn: Cantidad de kilobytes escritos:
- kB_dscd:

A continuacion, se muestra un ejemplo del uso de iostat.

```
gmovia@gmovia-System-Product-Name:~/Escritorio/Proyectos/PerformanceTP$ iostat
Linux 5.15.0-48-generic (gmovia-System-Product-Name) 08/10/22 _x86_64_ (12 CPU)
```

avg-cpu:	%user	%nice	%system	%iowait	%steal	%idle			
	1,58	0,01	0,36	0,59	0,00	97,47			
Device	tps	kB_read/s	kB_wrtn/s	kB_dscd/s	kB_read	kB_wrtn	kB_dscd		
loop0	0,00	0,00	0,00	0,00	17	0	0		
loop1	0,00	0,02	0,00	0,00	350	0	0		
loop10	0,00	0,07	0,00	0,00	1064	0	0		
loop11	0,13	0,62	0,00	0,00	10078	0	0		
loop12	0,00	0,02	0,00	0,00	367	0	0		
loop13	0,00	0,07	0,00	0,00	1081	0	0		
loop14	0,00	0,07	0,00	0,00	1134	0	0		
loop15	0,00	0,02	0,00	0,00	351	0	0		
loop16	0,06	0,65	0,00	0,00	10567	0	0		
loop17	0,00	0,02	0,00	0,00	348	0	0		
loop18	0,06	2,10	0,00	0,00	34275	0	0		
loop19	0,00	0,00	0,00	0,00	54	0	0		
loop2	0,00	0,02	0,00	0,00	359	0	0		
loop20	0,00	0,01	0,00	0,00	161	0	0		
loop21	0,00	0,00	0,00	0,00	14	0	0		
loop3	0,00	0,02	0,00	0,00	350	0	0		
loop4	0,04	0,41	0,00	0,00	6629	0	0		
loop5	0,00	0,07	0,00	0,00	1072	0	0		
loop6	0,00	0,07	0,00	0,00	1081	0	0		
loop7	0,00	0,07	0,00	0,00	1073	0	0		
loop8	0,00	0,07	0,00	0,00	1071	0	0		
loop9	0,11	1,16	0,00	0,00	18959	0	0		
sda	13,81	120,36	178,79	0,00	1963932	2917301	0		

Figura 17: Ejemplo de uso de iostat.

iotop El comando iotop es similar al comando top, pero en este se muestra la actividad del disco en tiempo real. Esta utilidad observa la información de uso de E / S del kernel y muestra una tabla del uso actual de E / S a través de procesos o subprocesos en el sistema. También muestra el ancho de banda y el tiempo de E / S leído y escrito de cada proceso o subproceso.

Las columnas que nos brinda el comando iotop son las siguientes:

- TID: Identificador del proceso.
- PRIO:
- USER: Usuario que ejecuta el proceso.
- DISK READ: Cantidad de bytes leídos por segundo.
- DISK WRITE: Cantidad de bytes escritos por segundo.
- SHAWPIN
- COMMAND: Nombre de la tarea.

A continuacion, se muestra un ejemplo del uso de iotop.

Total DISK READ:			0.00 B/s	Total DISK WRITE:			0.00 B/s
Current DISK READ:			0.00 B/s	Current DISK WRITE:			0.00 B/s
TID	PRIO	USER	DISK READ	DISK WRITE	SWAPIN	IO>	COMMAND
1	be/4	root	0.00 B/s	0.00 B/s	?unavailable?		init splash
2	be/4	root	0.00 B/s	0.00 B/s	?unavailable?		[kthreadd]
3	be/4	root	0.00 B/s	0.00 B/s	?unavailable?		[rcu_gp]
4	be/4	root	0.00 B/s	0.00 B/s	?unavailable?		[rcu_par_gp]
5	be/4	root	0.00 B/s	0.00 B/s	?unavailable?		[netns]
7	be/4	root	0.00 B/s	0.00 B/s	?unavailable?		[kworker/0:0H-events_highpri]
9	be/4	root	0.00 B/s	0.00 B/s	?unavailable?		[kworker/0:1H-kblockd]
10	be/4	root	0.00 B/s	0.00 B/s	?unavailable?		[mm_percpu_wq]
11	be/4	root	0.00 B/s	0.00 B/s	?unavailable?		[rcu_tasks_rude_]
12	be/4	root	0.00 B/s	0.00 B/s	?unavailable?		[rcu_tasks_trace]
13	be/4	root	0.00 B/s	0.00 B/s	?unavailable?		[ksoftirqd/0]
14	be/4	root	0.00 B/s	0.00 B/s	?unavailable?		[rcu_sched]
15	be/4	root	0.00 B/s	0.00 B/s	?unavailable?		[migration/0]
16	be/4	root	0.00 B/s	0.00 B/s	?unavailable?		[idle_inject/0]
18	be/4	root	0.00 B/s	0.00 B/s	?unavailable?		[cpuhp/0]
19	be/4	root	0.00 B/s	0.00 B/s	?unavailable?		[cpuhp/1]
20	be/4	root	0.00 B/s	0.00 B/s	?unavailable?		[idle_inject/1]
21	be/4	root	0.00 B/s	0.00 B/s	?unavailable?		[migration/1]
22	be/4	root	0.00 B/s	0.00 B/s	?unavailable?		[ksoftirqd/1]
24	be/4	root	0.00 B/s	0.00 B/s	?unavailable?		[kworker/1:0H-events_highpri]
25	be/4	root	0.00 B/s	0.00 B/s	?unavailable?		[cpuhp/2]
26	be/4	root	0.00 B/s	0.00 B/s	?unavailable?		[idle_inject/2]
27	be/4	root	0.00 B/s	0.00 B/s	?unavailable?		[migration/2]
28	be/4	root	0.00 B/s	0.00 B/s	?unavailable?		[ksoftirqd/2]
30	be/4	root	0.00 B/s	0.00 B/s	?unavailable?		[kworker/2:0H-events_highpri]
31	be/4	root	0.00 B/s	0.00 B/s	?unavailable?		[cpuhp/3]
32	be/4	root	0.00 B/s	0.00 B/s	?unavailable?		[idle_inject/3]
33	be/4	root	0.00 B/s	0.00 B/s	?unavailable?		[migration/3]
34	be/4	root	0.00 B/s	0.00 B/s	?unavailable?		[ksoftirqd/3]
36	be/4	root	0.00 B/s	0.00 B/s	?unavailable?		[kworker/3:0H-events_highpri]
37	be/4	root	0.00 B/s	0.00 B/s	?unavailable?		[cpuhp/4]
38	be/4	root	0.00 B/s	0.00 B/s	?unavailable?		[idle_inject/4]

Figura 18: Ejemplo de uso de iotop.

2.3.2.3. Analisis

El programa realiza múltiples escrituras de disco, ya que por cada ciclo del loop, tendrá su propio ciclo en el cual llamará a la syscall `write()`, la cual permitirá realizar la escritura.

Antes de la ejecución del programa, veamos como es el estado de los dispositivos de entrada y salida. Para ello utilizaremos el comando filtrando por dispositivo, ya que solamente tendremos cuenta `sda`, el cual representa al disco duro.

```
gmovia@gmovia-System-Product-Name:~$ iostat sda 2 5
Linux 5.15.0-48-generic (gmovia-System-Product-Name) 08/10/22 _x86_64_ (12 CPU)

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           1,92    0,01    0,45    0,57    0,00   97,06

Device            tps    kB_read/s    kB_wrtn/s    kB_dscd/s    kB_read    kB_wrtn    kB_dscd
sda              14,24    117,66    194,55         0,00   2162853   3576045         0

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           2,60    0,00    0,46    0,00    0,00   96,94

Device            tps    kB_read/s    kB_wrtn/s    kB_dscd/s    kB_read    kB_wrtn    kB_dscd
sda              0,00         0,00         0,00         0,00         0         0         0

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           2,88    0,00    0,50    0,08    0,00   96,53

Device            tps    kB_read/s    kB_wrtn/s    kB_dscd/s    kB_read    kB_wrtn    kB_dscd
sda             51,00         0,00    306,00         0,00         0     612         0

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           3,29    0,00    0,29    0,00    0,00   96,42

Device            tps    kB_read/s    kB_wrtn/s    kB_dscd/s    kB_read    kB_wrtn    kB_dscd
sda              0,50         0,00         8,00         0,00         0        16         0

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           2,60    0,00    0,46    0,00    0,00   96,94

Device            tps    kB_read/s    kB_wrtn/s    kB_dscd/s    kB_read    kB_wrtn    kB_dscd
sda             15,00         0,00    98,00         0,00         0     196         0
```

Figura 19: Salida de `iostat` antes de la ejecución del programa, realizando cinco iteraciones, una cada dos segundos.

Ejecutando el programa y repitiendo el procedimiento, obtenemos los siguientes resultados.

```

gmovia@gmovia-System-Product-Name:~$ iostat sda 2 5
Linux 5.15.0-48-generic (gmovia-System-Product-Name)      08/10/22      _x86_64_      (12 CPU)

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           1,95    0,01    0,45    0,57    0,00   97,01

Device            tps    kB_read/s    kB_wrtn/s    kB_dscd/s    kB_read    kB_wrtn    kB_dscd
sda              14,38      117,22      196,52         0,00    2177969    3651445         0

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           2,68    0,00    0,50    7,99    0,00   88,83

Device            tps    kB_read/s    kB_wrtn/s    kB_dscd/s    kB_read    kB_wrtn    kB_dscd
sda             129,00         2,00     1324,00         0,00         4      2648         0

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           5,87    0,00    1,01    7,88    0,00   85,25

Device            tps    kB_read/s    kB_wrtn/s    kB_dscd/s    kB_read    kB_wrtn    kB_dscd
sda             127,00         0,00     1276,00         0,00         0      2552         0

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           3,96    0,04    0,50    7,88    0,00   87,62

Device            tps    kB_read/s    kB_wrtn/s    kB_dscd/s    kB_read    kB_wrtn    kB_dscd
sda             115,50         0,00     1238,00         0,00         0      2476         0

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           2,92    0,00    0,50    8,39    0,00   88,19

Device            tps    kB_read/s    kB_wrtn/s    kB_dscd/s    kB_read    kB_wrtn    kB_dscd
sda             120,00         0,00     1244,00         0,00         0      2488         0

```

Figura 20: Salida de iostat durante la ejecución del programa, realizando cinco iteraciones, una cada dos segundos.

Podemos notar que efectivamente la cantidad de bytes escritos aumenta considerablemente, y esto se debe a las escrituras realizadas. A su vez, también aumenta la variable tps, y esto se debe a que se realiza una mayor cantidad de solicitudes al disco duro. Validamos nuevamente esta información utilizando el comando iotop.

Total DISK READ:		0.00 B/s		Total DISK WRITE:		310.61 K/s	
Current DISK READ:		0.00 B/s		Current DISK WRITE:		666.51 K/s	
TID	PRI	USER	DISK READ	DISK WRITE	SWAPIN	IO>	COMMAND
338	be/3	root	0.00 B/s	45.30 K/s	?unavailable?		[jbd2/sda4-8]
2111	be/4	mongodb	0.00 B/s	16.18 K/s	?unavailable?		mongod --config /etc/mongod
5230	be/4	gmovia	0.00 B/s	3.24 K/s	?unavailable?		chrome --enable-crashpad
30760	be/4	gmovia	0.00 B/s	6.47 K/s	?unavailable?		chrome --type=utility --uti
32148	be/4	gmovia	0.00 B/s	239.43 K/s	?unavailable?		./a.out
1	be/4	root	0.00 B/s	0.00 B/s	?unavailable?		init splash

Figura 21: Salida de iotop durante la ejecución del programa.

El comando nos brinda información en vivo acerca de las lecturas y escrituras que realiza el programa. En este caso, se puede observar que en la anteúltima columna, el programa (a.out) está escribiendo en disco a una velocidad de 239 Kb/s.

2.3.3. Conclusión

El bucle infinito realiza varias escrituras al disco por ciclo, de forma tal que se generan una gran cantidad de solicitudes al dispositivo. El uso de los comandos iostat y iotop nos

permitió obtener información acerca de la cantidad de kilobytes escritos por segundo, junto con otras estadísticas relevantes, que permitieron obtener un panorama más completo del estado del disco.

3. Conclusión

El uso de las herramientas proveídas por Linux fueron claves para el análisis de la performance de cada uno de los programas estudiados. Podemos concluir que se destaca la utilización de las herramientas `top` y `iostat` por sobre las demás, ya que permiten obtener una estadística (del CPU en caso de `top`, y del disco en caso de `iostat`) en vivo del consumo de los recursos por parte de cada uno de los procesos que se encuentran ejecutados.