

Taller 5 – Máquina virtual
Sistemas Operativos



Daniel Felipe Castro Moreno

Profesor:

John Corredor Franco

Dpto. de Ingeniería de Sistemas

Pontificia Universidad Javeriana

Bogotá D.C.

24 de octubre del 2024

Informe de laboratorio

La presente máquina virtual tiene las siguientes características:

- **Procesadores:** 4 núcleos, cada uno con un hilo y un socket.
- **Memoria caché:**
 - L1d: 4 instancias de 128 KiB cada una.
 - L1i: 4 instancias de 128 KiB cada una.
 - L2: 4 instancias de 4 MiB cada una.
 - L3: 4 instancias de 143 MiB cada una.
- **Memoria RAM:** 11.673 MiB.

Esta configuración sigue la jerarquía de memoria, donde los niveles más cercanos al procesador son más rápidos, pero con menor capacidad de almacenamiento.

- **Arquitectura:** Soporte para 32 y 64 bits, con direccionamiento de 43 bits para la memoria física y 48 bits para la memoria virtual. El procesador utilizado es un Intel(R) Xeon(R) Gold 6240R CPU @ 2.40GHz.
- **Almacenamiento persistente:** 21 GB.

Dadas las anteriores características, la máquina virtual presenta características que permiten el desarrollo de pruebas. No obstante, la poca cantidad de memoria RAM y de memoria persistente resulta limitante para el procesamiento de gran cantidad de datos o aplicaciones de alto rendimiento.

A continuación, se presentan algunos de los comandos utilizados durante el laboratorio:

```

top - 10:25:45 up 98 days, 20:17, 1 user, load average: 0.00, 0.00, 0.00
Tasks: 250 total, 1 running, 249 sleeping, 0 stopped, 0 zombie
%Cpu(s): 0.2 us, 0.2 sy, 0.0 ni, 99.4 id, 0.0 wa, 0.1 hi, 0.1 si, 0.0 st
MiB Mem : 11673.0 total, 4679.4 free, 2745.6 used, 4598.3 buff/cache
MiB Swap: 4100.0 total, 4100.0 free, 0.0 used, 8927.4 avail Mem

```

	PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
	3779350	root	20	0	2622556	311928	102584	S	1.3	2.6	431:21.39	ampdaemon
	176172	zabbix	20	0	16420	2372	1536	S	0.3	0.0	34:08.27	zabbix_agentd
	320105	root	20	0	0	0	0	I	0.3	0.0	0:01.01	kworker/0:3-events
	323120	estudia+	20	0	10716	4352	3456	R	0.3	0.0	0:00.26	top
	1	root	20	0	175024	18892	10940	S	0.0	0.2	1:38.28	systemd
	2	root	20	0	0	0	0	S	0.0	0.0	0:03.08	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	slub_flushwq
	6	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	netns
	8	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	kworker/0:0H-events_highpri
	10	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	mm_percpu_wq
	12	root	20	0	0	0	0	I	0.0	0.0	0:00.00	rcu_tasks_kthre
	13	root	20	0	0	0	0	I	0.0	0.0	0:00.00	rcu_tasks_rude
	14	root	20	0	0	0	0	I	0.0	0.0	0:00.00	rcu_tasks_trace
	15	root	20	0	0	0	0	S	0.0	0.0	0:05.85	ksoftirqd/0
	16	root	20	0	0	0	0	S	0.0	0.0	8:08.84	pr/tty0
	17	root	20	0	0	0	0	I	0.0	0.0	19:15.05	rcu_preempt
	18	root	rt	0	0	0	0	S	0.0	0.0	0:21.26	migration/0
	19	root	-51	0	0	0	0	S	0.0	0.0	0:00.00	idle_inject/0
	21	root	20	0	0	0	0	S	0.0	0.0	0:00.00	cpuhp/0

Figura 1. top.

```

[estudiante@ING-PDGE27 ~]$ lscpu
Architecture: x86_64
CPU op-mode(s): 32-bit, 64-bit
Address sizes: 43 bits physical, 48 bits virtual
Byte Order: Little Endian
CPU(s): 4
On-line CPU(s) list: 0-3
Vendor ID: GenuineIntel
Model name: Intel(R) Xeon(R) Gold 6240R CPU @ 2.40GHz
CPU family: 6
Model: 85
Thread(s) per core: 1
Core(s) per socket: 1
Socket(s): 4
Stepping: 7
BogoMIPS: 4788.74
Flags: fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush mmx fxsr sse sse2 ss syscall nx pdpe1gb rdtscp lm constant_tsc
arch_perfmon nopl xtopology tsc_reliable nonstop_tsc cpuid tsc_known_freq pni pclmulqdq ssse3 fma cx16 pcid sse4_1 sse4_2 x2apic movbe popcnt t
sc deadline_timer aes xsave avx f16c rdrand hypervisor lahf_lm abm 3dnowprefetch ssbd ibrs ibpb stibp ibrs_enhanced fsgsbase tsc_adjust bmi1 av
x2 smap bmi2 invpcid avx512f avx512dq rdseed adx smap clflushopt clwb avx512cd avx512bw avx512vl xsaveopt xsavec xsaves arat pku ospke md_clear
flush_lld arch_capabilities

Virtualization features:
Hypervisor vendor: VMware
Virtualization type: full

Caches (sum of all):
L1d: 128 KiB (4 instances)
L1i: 128 KiB (4 instances)
L2: 4 MiB (4 instances)
L3: 143 MiB (4 instances)

NUMA:
NUMA node(s): 1
NUMA node0 CPU(s): 0-3

Vulnerabilities:
Gather data sampling: Unknown: Dependent on hypervisor status
Itlb multihit: KVM: Mitigation: VMX unsupported
L1tf: Not affected
Mds: Not affected
Meltdown: Not affected
Mmio stale data: Vulnerable: Clear CPU buffers attempted, no microcode; SMT Host state unknown
Retbleed: Mitigation; Enhanced IBRS
Spec rstack overflow: Not affected
Spec store bypass: Mitigation; Speculative Store Bypass disabled via prctl
Spectre v1: Mitigation; usercopy/swapgs barriers and __user pointer sanitization
Spectre v2: Mitigation; Enhanced / Automatic IBRS, IBPB conditional, RSB filling, PBRSE-eIBRS SW sequence

```

Figura 2. lscpu.

```
[estudiante@ING-PDGE27 ~]$ df -h
Filesystem                Size      Used Avail Use% Mounted on
devtmpfs                   4.0M         0  4.0M   0% /dev
tmpfs                      5.7G         0  5.7G   0% /dev/shm
tmpfs                      2.3G      37M   2.3G   2% /run
/dev/mapper/rl_plantillarocky9-root 24G    3.3G    21G  14% /
/dev/sda2                  960M    389M   572M  41% /boot
/dev/sda1                 1022M     7.1M  1015M   1% /boot/efi
/dev/mapper/rl_plantillarocky9-var  15G    1.4G    14G   9% /var
/dev/mapper/rl_plantillarocky9-home  15G   140M    15G   1% /home
tmpfs                      1.2G     4.0K    1.2G   1% /run/user/1001
[estudiante@ING-PDGE27 ~]$
```

Figura 3. df - h.

Actividad:

1. Se crea una carpeta.

```
$mkdir CastroD
$ls
```

```
[estudiante@ING-PDGE27 ~]$ mkdir CastroD
[estudiante@ING-PDGE27 ~]$ ls
CastroD Trejos
```

Figura 4. Creación de la primera carpeta.

2. Se entra a la carpeta creada.

```
$cd CastroD
$ls
```

```
[estudiante@ING-PDGE27 ~]$ cd CastroD
[estudiante@ING-PDGE27 CastroD]$
```

Figura 5. Entrada a la primera carpeta.

3. Se observa la ruta actual.

```
$pwd
```

```
[estudiante@ING-PDGE27 CastroD]$ pwd
/home/estudiante/CastroD
```

Figura 6. Ruta actual.

4. Se crea un nuevo directorio.

\$mkdir fork

\$cd fork

```
[estudiante@ING-PDGE27 CastroD]$ mkdir fork
[estudiante@ING-PDGE27 CastroD]$ cd fork
```

Figura 7. Creación de segundo directorio.

5. Creación de 2 ficheros.

\$nano cliente.c

\$ nano servidor.c

```
GNU nano 5.6.1 cliente.c Modified
Fecha: 24/10/2024
Autores: Daniel Castro, María Paula Rodríguez, Eliana Cepeda, Daniel Gómez
Materia: Sistemas Operativos
Tema: Taller 4 - PipeName Bidireccional

Descripción:
Este programa permite la comunicación bidireccional de un cliente a un servidor usando
NamePipes y un archivo FIFO.
*****
CLIENTE
*****/

#include <stdio.h> // Para entradas y salidas estándar
#include <sys/stat.h> // Para gestionar propiedades de archivos
#include <sys/types.h> // Para definir tipos de datos como pid_t
#include <fcntl.h> // Para controlar archivos
#include <unistd.h> // Para llamadas al sistema como fork()
#include <string.h> // Para manejo de cadenas de caracteres

#define FIFO_FILE "/tmp/fifo_twoway" // Definición del nombre del archivo FIFO

int main() {
    int fd; // Descriptor para el archivo FIFO
    int end_process; // Indicador para terminar el proceso
    int stringlen; // Longitud de la cadena de entrada
    int read_bytes; // Lector de bytes
    char readbuf[80]; // Buffer de 80 caracteres para lectura de la entrada
    char end_str[5]; // Cadena "end" para finalizar
    printf("FIFO CLIENT: Send messages, infinitely, to end enter \"end\\n\""); // Muestra mensaje inicial
    fd = open(FIFO_FILE, O_CREAT|O_RDWR); // Abre el archivo FIFO en modo escritura y lectura
    strcpy(end_str, "end"); // Inicializa la cadena de terminación

    while (1) { // Ciclo infinito para enviar mensajes
        printf("Enter string: "); // Solicita entrada
        fgets(readbuf, sizeof(readbuf), stdin); // Lee la entrada del usuario
        stringlen = strlen(readbuf); // Calcula la longitud de la cadena
        readbuf[stringlen - 1] = '\\0'; // Ingresa el caracter de final
        end_process = strcmp(readbuf, end_str); // Compara la entrada con "end"
    }

    return 0;
}
```

Figura 8. Contenido de cliente.c.

```

GNU nano 5.6.1                                servidor.c                                Modified
/*****
Fecha: 24/10/2024
Autores: Daniel Castro, María Paula Rodríguez, Eliana Cepeda, Daniel Gómez
Materia: Sistemas Operativos
Tema: Taller 4 - PipeName Bidireccional

Descripción:
    Este programa permite la comunicación bidireccional de un cliente a un servidor usando
    NamePipes y un archivo FIFO.
*****/

SERVIDOR
*****/

#include <stdio.h> // Para entradas y salidas estándar
#include <sys/stat.h> // Para gestionar propiedades de archivos
#include <sys/types.h> // Para definir tipos de datos como pid_t
#include <fcntl.h> // Para controlar archivos
#include <unistd.h> // Para llamadas al sistema como fork()
#include <string.h> // Para manejo de cadenas de caracteres

#define FIFO_FILE "/tmp/fifo_twoway" // Definición del nombre del archivo FIFO

/* Función principal del servidor */
void reverse_string(char *);
int main() {
    int fd; // Descriptor para el archivo FIFO
    char readbuf[80]; // Buffer para leer los mensajes del cliente
    char end[10]; // Cadena para detectar el fin de la comunicación
    int to_end; // Indicador de fin de proceso
    int read_bytes; // Almacena el número de bytes leídos

    /* Crear FIFO si este no existe */
    mkfifo(FIFO_FILE, S_IFIFO|0660); // Crea un FIFO con permisos de lectura (2) y escritura (4)
    strcpy(end, "end"); // Inicializa la cadena que finaliza la comunicación
    fd = open(FIFO_FILE, O_RDONLY); // Abre el FIFO en modo lectura
    while(1) {
        read_bytes = read(fd, readbuf, sizeof(readbuf)); // Lee los datos del cliente
        readbuf[read_bytes] = '\0'; // Termina la cadena leída con '\0'
        printf("FIFO SERVER: Received string: \"%s\" and length is %d\n", readbuf, (int)strlen(readbuf)); // Imprime el mensaje recibido y su longitud
        to_end = strcmp(readbuf, end); // Compara la cadena recibida con la de cierre
    }
}

```

Figura 9. Contenido de servidor.c.

```

compilation terminated.
estudiante@ING-PDGE27 fork]$ gcc -o programa servidor.c
estudiante@ING-PDGE27 fork]$ ./programa
FIFO SERVER: Received string: "salir" and length is 5
FIFO SERVER: Sending Reversed String: "rilas" and length is 5
FIFO SERVER: Received string: "a" and length is 1
FIFO SERVER: Sending Reversed String: "a" and length is 1
FIFO SERVER: Received string: "Prueba" and length is 6
FIFO SERVER: Sending Reversed String: "abeurP" and length is 6
FIFO SERVER: Received string: "end" and length is 3
estudiante@ING-PDGE27 fork]$

[estudiante@ING-PDGE27 fork]$ ./cliente
FIFO_CLIENT: Send messages, infinitely, to end enter "end"
Enter string: Prueba
FIFOCLIENT: Sent string: "Prueba" and string length is 6
FIFOCLIENT: Received string: "abeurP" and length is 6
Enter string: salir
FIFOCLIENT: Sent string: "salir" and string length is 5
FIFOCLIENT: Received string: "salir" and length is 5
Enter string: end
FIFOCLIENT: Sent string: "end" and string length is 3
[estudiante@ING-PDGE27 fork]$

```

Figura 10. Funcionamiento de ambos procesos.

6. Salir de la carpeta actual.

\$cd

```

[estudiante@ING-PDGE27 fork]$ cd
[estudiante@ING-PDGE27 ~]$

```

Figura 11. Salida de la carpeta actual.

7. Crear carpeta.

\$mkdir posix

\$cd posix

```

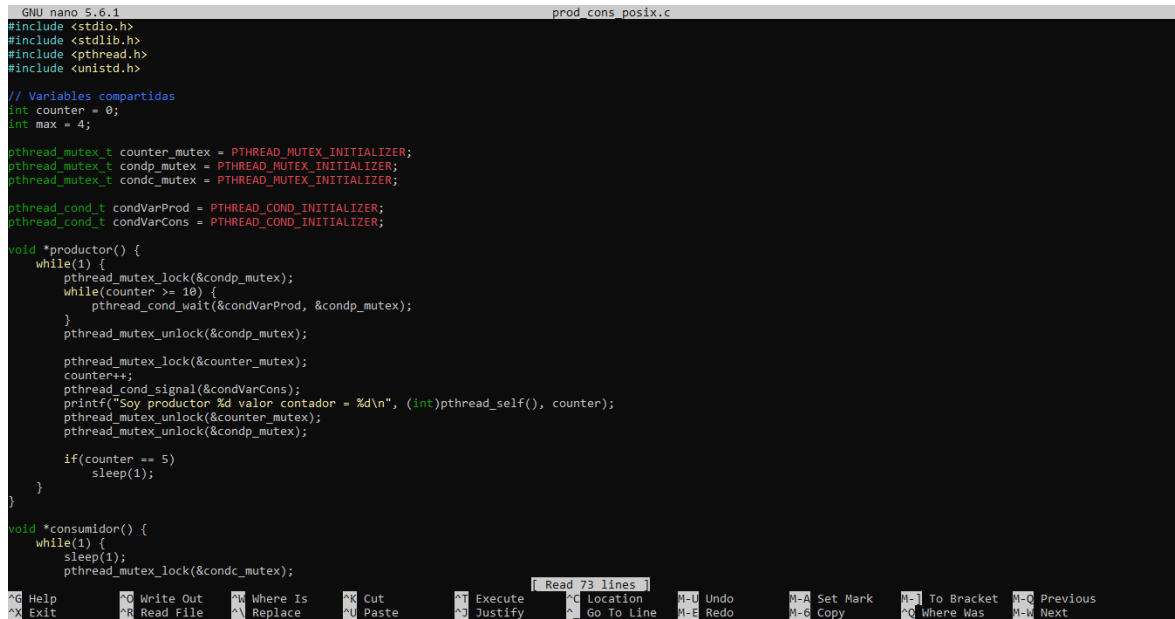
[estudiante@ING-PDGE27 ~]$ cd CastroD/posix
[estudiante@ING-PDGE27 posix]$

```

Figura 12. Creación de la tercera carpeta.

8. Crear archivo.

\$nano prod_cons_posix.c



```
GNU nano 5.6.1 prod_cons_posix.c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <unistd.h>

// Variables compartidas
int counter = 0;
int max = 4;

pthread_mutex_t counter_mutex = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t condp_mutex = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t condc_mutex = PTHREAD_MUTEX_INITIALIZER;

pthread_cond_t condVarProd = PTHREAD_COND_INITIALIZER;
pthread_cond_t condVarCons = PTHREAD_COND_INITIALIZER;

void *producer() {
    while(1) {
        pthread_mutex_lock(&condp_mutex);
        while(counter >= 10) {
            pthread_cond_wait(&condVarProd, &condp_mutex);
        }
        pthread_mutex_unlock(&condp_mutex);

        pthread_mutex_lock(&counter_mutex);
        counter++;
        pthread_cond_signal(&condVarCons);
        printf("Soy productor %d valor contador = %d\n", (int)pthread_self(), counter);
        pthread_mutex_unlock(&counter_mutex);
        pthread_mutex_unlock(&condp_mutex);

        if(counter == 5)
            sleep(1);
    }
}

void *consumidor() {
    while(1) {
        sleep(1);
        pthread_mutex_lock(&condc_mutex);
```

Figura 13. Contenido de prod_cons_posix.c.

```
[estudiante@ING-PDGE27 posix]$ gcc -o pc prod_cons_posix.c
[estudiante@ING-PDGE27 posix]$ ./pc
Soy productor 2023749184 valor contador = 1
Soy productor 2023749184 valor contador = 2
Soy productor 2023749184 valor contador = 3
Soy productor 2023749184 valor contador = 4
Soy productor 2023749184 valor contador = 5
Soy productor 2006963776 valor contador = 6
Soy productor 2006963776 valor contador = 7
Soy productor 2006963776 valor contador = 8
Soy productor 2006963776 valor contador = 9
Soy productor 2006963776 valor contador = 10
Soy consumidor 2015356480 valor contador = 10
Soy productor 2006963776 valor contador = 10
Soy consumidor 1879045696 valor contador = 10
Soy consumidor 1990178368 valor contador = 9
Soy productor 1981785664 valor contador = 9
Soy productor 1981785664 valor contador = 10
Soy productor 1998571072 valor contador = 11
Soy consumidor 1973392960 valor contador = 11
Soy consumidor 1956607552 valor contador = 10
Soy productor 2006963776 valor contador = 10
Soy consumidor 2015356480 valor contador = 10
Soy productor 2023749184 valor contador = 10
Soy consumidor 1990178368 valor contador = 10
Soy consumidor 1879045696 valor contador = 9
Soy productor 1998571072 valor contador = 9
Soy productor 1981785664 valor contador = 10
Soy consumidor 1973392960 valor contador = 10
```

Figura 14. Ejecución de prod_cons_posix.c.

Enlace al repositorio:

<https://github.com/Dani2044/Sistemas-Operativos/tree/main/Taller%205%20-%20M%C3%A1quina%20Virtual>