

# M6 Lab: Process Control

---

CITA 171: OPERATING SYSTEM USE & ADMIN

Marco Gonzalez

SUNY CANTON | 34 CORNELL DRIVE, CANTON, NEW YORK 13617

## TABLE OF CONTENTS

---

1	Table of Figures .....	1
2	Preparations.....	1
3	Process .....	2
4	Process Information.....	2
5	Real-Time Process Monitoring.....	3
6	Bash Job Management.....	4
6.1	Creating Bash Jobs .....	4
6.2	Displaying Bash Jobs .....	5
6.3	Switching Between Bash Jobs .....	5
6.4	Foreground and Background Processes.....	5
7	Kill Command .....	6

## 1 TABLE OF FIGURES

---

Figure 1. Process Information Using the ps Command.....	2
Figure 2. Child-Parent Process Relationship Using pstree Command .....	3
Figure 3. The top Command .....	3
Figure 4. Starting and Stopping Two Bash Jobs .....	4
Figure 5. The jobs Command .....	5
Figure 6. Starting and Terminating a Foreground Process .....	6
Figure 7. Starting and Failing to Terminate a Background Process .....	6
Figure 8. Kill Signals.....	7
Figure 9. The SIGSTOP Signal .....	8
Figure 10. SIGCONT and SIGKILL Signals .....	8

## 2 PREPARATIONS

---

Start the CITA 171 VM and log in. Open a terminal window.

### 3 PROCESS

A **process** is an abstraction of program execution. A (computer) **program** is a list of instructions for a computer to perform tasks.

### 4 PROCESS INFORMATION

The **ps** command with the **-ef** or the **aux** option displays a snapshot of the current process execution state. See Figure 1. In this example, the first ten (10) lines of the **ps** command output are shown.

```
(04/24 19:28:36) cital71@cital71-vm: ~
$ ps -ef | head
UID          PID    PPID  C STIME TTY          TIME CMD
root          1        0  0  18:51 ?        00:00:03 /sbin/init splash
root          2        0  0  18:51 ?        00:00:00 [kthreadd]
root          3        2  0  18:51 ?        00:00:00 [rcu_gp]
root          4        2  0  18:51 ?        00:00:00 [rcu_par_gp]
root          5        2  0  18:51 ?        00:00:00 [slub_flushwq]
root          6        2  0  18:51 ?        00:00:00 [netns]
root          8        2  0  18:51 ?        00:00:00 [kworker/0:0H-events_highpri]
root          9        2  0  18:51 ?        00:00:00 [kworker/u2:0-events_unbound]
root         10        2  0  18:51 ?        00:00:00 [mm_percpu_wq]
(04/24 19:32:13) cital71@cital71-vm: ~
$
```

Figure 1. Process Information Using the **ps** Command

Note the **UID**, **PID**, **PPID**, and **CMD** columns. The **UID** column shows the **process owner** user IDs. The process owner of a process is the user who can manage the process besides the system administrators. The **PID** column shows the process IDs. Each process is assigned a unique number that is used to identify the process. The **PPID** column shows the processes' **parent process** IDs. A process is created from another process that the created process becomes the **child process** of the process that created the process.

The child-parent relationship can be viewed using the **pstree** command. See Figure 2. In this example, the first ten (10) lines of the **pstree** command output are shown. Note that the number in the parentheses is the **PID**. **PID 1** is always a process called **systemd**, which is the mother of all the other processes.

```

(04/24 19:52:33) cita171@cita171-vm: ~
$ pstree -p | head
systemd(1)-+-ModemManager(667)-+-{ModemManager}(699)
|                                     \- {ModemManager}(702)
|   -NetworkManager(589)-+-{NetworkManager}(651)
|                       \- {NetworkManager}(653)
|   -VBoxClient(1915)---VBoxClient(1916)-+-{VBoxClient}(1917)
|                                   \- {VBoxClient}(1918)
|   -VBoxClient(1927)---VBoxClient(1928)-+-{VBoxClient}(1929)
|                                   \- {VBoxClient}(1930)
|   -VBoxClient(1934)---VBoxClient(1935)-+-{VBoxClient}(1936)
|                                   \- {VBoxClient}(1937)
(04/24 19:52:43) cita171@cita171-vm: ~
$
  
```

Figure 2. Child-Parent Process Relationship Using pstree Command

## 5 REAL-TIME PROCESS MONITORING

The **top** command shows the current process activities. See Figure 3. Press the **Q** key to exit.

```

top - 20:11:31 up 1:20, 1 user, load average: 0.00, 0.00, 0.07
Tasks: 185 total, 1 running, 184 sleeping, 0 stopped, 0 zombie
Cpu(s): 2.0 us, 1.7 sy, 0.0 ni, 96.2 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
Mem: 1076.1 total, 348.0 free, 728.8 used, 798.4 buff/cache
Mem Swap: 1401.6 total, 1306.4 free, 95.2 used, 949.5 avail Mem
  
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
2001	cita171	20	0	3715332	278052	84276	S	2.3	13.7	0:26.43	gnome-shell
1798	cita171	20	0	284840	62224	29648	S	1.7	3.1	0:08.41	Xorg
2565	cita171	20	0	813776	41548	29164	S	1.3	2.1	0:02.63	gnome-terminal-
1935	cita171	20	0	155404	1704	1472	S	0.7	0.1	0:14.50	VBoxClient
589	root	20	0	262088	15360	12516	S	0.3	0.8	0:00.59	NetworkManager
29217	cita171	20	0	11864	3776	3260	R	0.3	0.2	0:00.01	top
1	root	20	0	169696	12660	7988	S	0.0	0.6	0:04.28	systemd
2	root	20	0	0	0	0	S	0.0	0.0	0:00.00	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
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	0:01.04	ksoftirqd/0
14	root	20	0	0	0	0	I	0.0	0.0	0:01.34	rcu_sched
15	root	rt	0	0	0	0	S	0.0	0.0	0:00.06	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	kdevtmpfs
20	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	inet_frag_wq
21	root	20	0	0	0	0	S	0.0	0.0	0:00.00	kauditd

Figure 3. The top Command

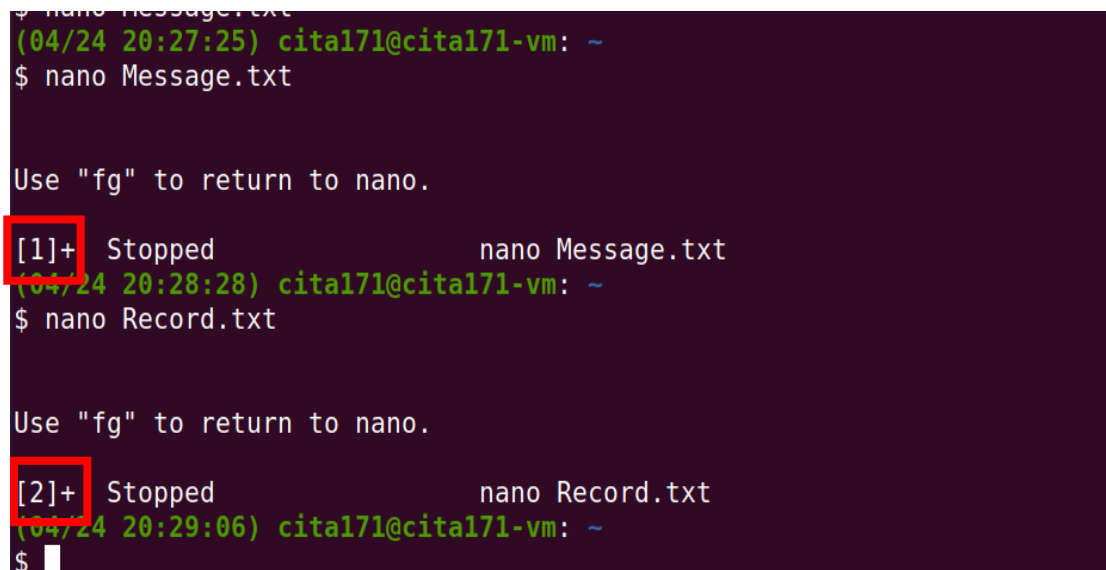
This command displays process and memory information, including **Tasks** (the number of processes), **Load Average** (1 minute, 5 minutes, and 15 minutes averages), user (**us**) process load, system (**sy**) process load, idle (**id**) load percentages. Usually, the higher the id load is, the better.

## 6 BASH JOB MANAGEMENT

### 6.1 CREATING BASH JOBS

Bash jobs are processes that are created from a Bash shell process. See Figure 4. In this example, two nano processes are created by following these steps:

1. Execute **nano Message.txt** to start a nano process.
2. Type *This is Message.txt*.
3. Hold down the **Ctrl** key and press the **Z** key (**Ctrl+Z**). This keyboard shortcut stops the nano process.
4. Execute **nano Record.txt** to start a nano process.
5. Type *This is Record.txt*.
6. Press **Ctrl+Z**.



```
$ nano Message.txt
(04/24 20:27:25) cita171@cita171-vm: ~
$ nano Message.txt

Use "fg" to return to nano.
[1]+  Stopped                  nano Message.txt
(04/24 20:28:28) cita171@cita171-vm: ~
$ nano Record.txt

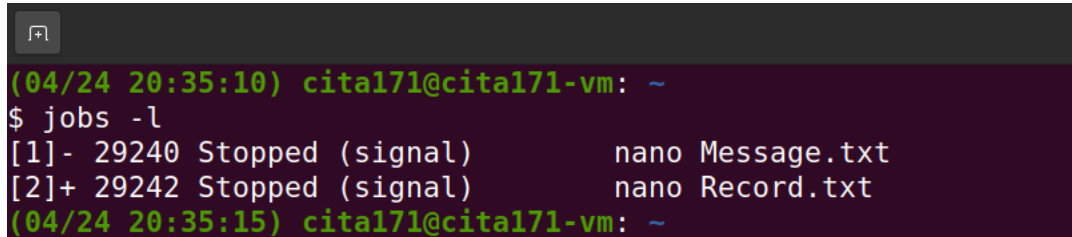
Use "fg" to return to nano.
[2]+  Stopped                  nano Record.txt
(04/24 20:29:06) cita171@cita171-vm: ~
$
```

Figure 4. Starting and Stopping Two Bash Jobs

The numbers in square brackets are **Bash job IDs**. Like PIDs, Bash keeps track of its jobs by assigning each one a unique number.

## 6.2 DISPLAYING BASH JOBS

The **jobs** command with the **-l** option shows the current Bash jobs and their corresponding Job IDs and PIDs. See Figure 5.



```
(04/24 20:35:10) cita171@cita171-vm: ~  
$ jobs -l  
[1]- 29240 Stopped (signal)      nano Message.txt  
[2]+ 29242 Stopped (signal)      nano Record.txt  
(04/24 20:35:15) cita171@cita171-vm: ~
```

Figure 5. The *jobs* Command

## 6.3 SWITCHING BETWEEN BASH JOBS

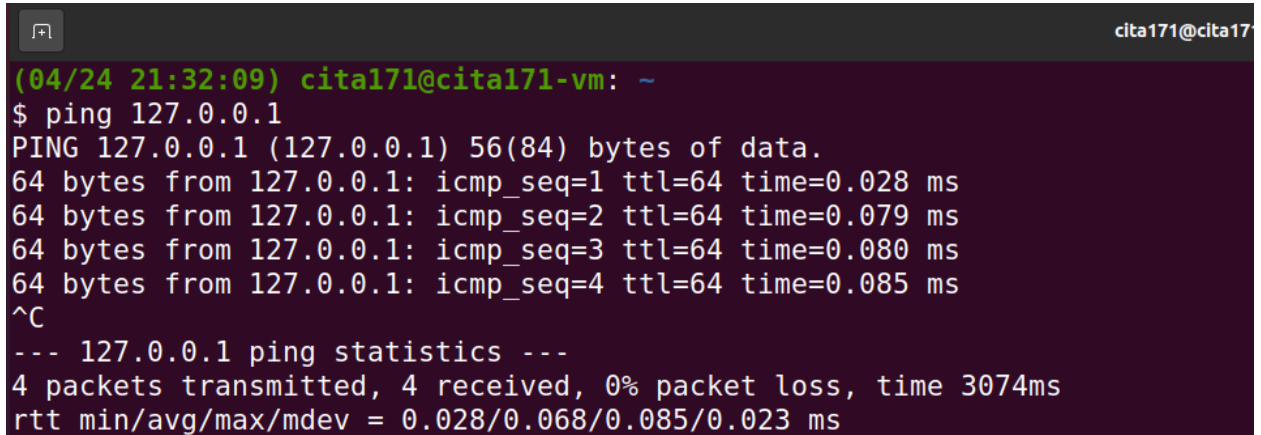
The user can switch one Bash job to another job by job ID. Follow these steps to switch between Job IDs 1 and 2.

1. Type **%1** and press Enter. It resumes the **nano Message.txt** job (Job ID 1).
2. Press **Ctrl+Z** to stop the job.
3. Type **%2** and press Enter. It resumes the **nano Record.txt** job (Job ID 2).
4. Press **Ctrl+Z** to stop the job.
5. Type **%1** and press Enter. It resumes the **nano Message.txt** job (Job ID 1).
6. Press **Ctrl+Z** to stop the job.

## 6.4 FOREGROUND AND BACKGROUND PROCESSES

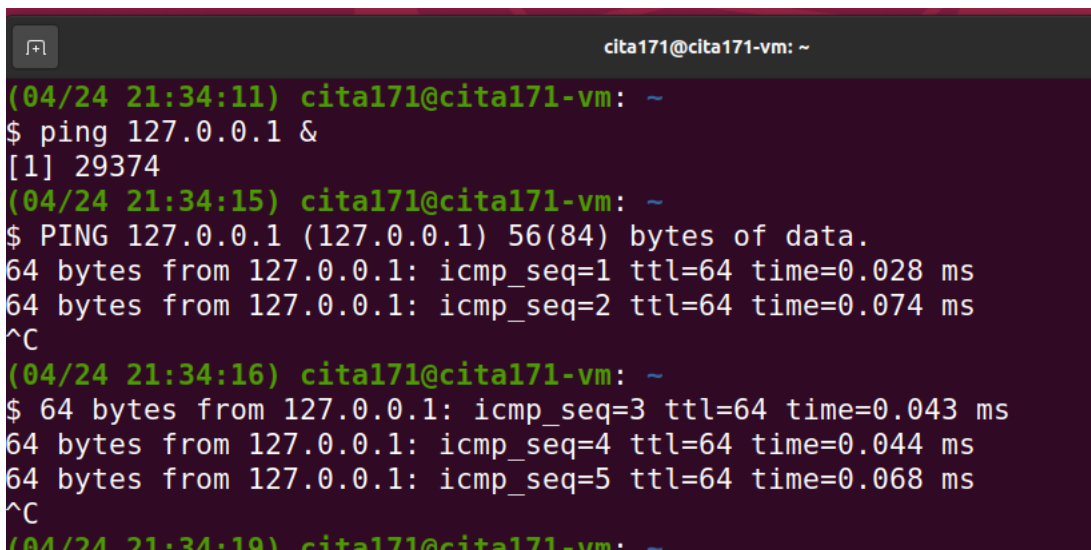
**Foreground processes** are the processes with which the user can interact directly. **Background processes** are the processes with which the user cannot interact directly. Follow these steps to test the differences. See also Figure 6 and Figure 7.

1. Type **ping 127.0.0.1** and press Enter. A ping process starts as a foreground process. This process continues forever. In other words, the command prompt is not returned, no matter how long the user waits.
2. Press **Ctrl+C**. The ping process terminates. Because this ping process was a foreground process, the user was able to interact with the process directly to terminate it.
3. Type **ping 127.0.0.1 &** and press Enter. (Note the ampersand character at the end.) A ping process starts as a background process.
4. Press **Ctrl+C**. The ping process does not terminate. Because this ping process is a background process, the user cannot directly interact with it to terminate it.
5. Close the Terminal program window.



```
(04/24 21:32:09) cita171@cita171-vm: ~  
$ ping 127.0.0.1  
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.  
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.028 ms  
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.079 ms  
64 bytes from 127.0.0.1: icmp_seq=3 ttl=64 time=0.080 ms  
64 bytes from 127.0.0.1: icmp_seq=4 ttl=64 time=0.085 ms  
^C  
--- 127.0.0.1 ping statistics ---  
4 packets transmitted, 4 received, 0% packet loss, time 3074ms  
rtt min/avg/max/mdev = 0.028/0.068/0.085/0.023 ms
```

Figure 6. Starting and Terminating a Foreground Process



```
(04/24 21:34:11) cita171@cita171-vm: ~  
$ ping 127.0.0.1 &  
[1] 29374  
(04/24 21:34:15) cita171@cita171-vm: ~  
$ PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.  
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.028 ms  
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.074 ms  
^C  
(04/24 21:34:16) cita171@cita171-vm: ~  
$ 64 bytes from 127.0.0.1: icmp_seq=3 ttl=64 time=0.043 ms  
64 bytes from 127.0.0.1: icmp_seq=4 ttl=64 time=0.044 ms  
64 bytes from 127.0.0.1: icmp_seq=5 ttl=64 time=0.068 ms  
^C  
(04/24 21:34:19) cita171@cita171-vm: ~
```

Figure 7. Starting and Failing to Terminate a Background Process

## 7 KILL COMMAND

The **kill** command is used to control processes. This command sends a special signal called a **kill signal** to processes, and they respond differently depending on the signal. When the command is used with the **-l** option, it lists 62 signals. See Figure 8.

```

(04/24 21:55:50) cita171@cita171-vm: ~
$ kill -l
 1) SIGHUP      2) SIGINT      3) SIGQUIT     4) SIGTILL     5) SIGTRAP
 6) SIGABRT     7) SIGBUS     8) SIGFPE      9) SIGKILL    10) SIGUSR1
11) SIGSEGV    12) SIGUSR2    13) SIGPTPE    14) SIGALRM    15) SIGTERM
16) SIGSTKFLT  17) SIGCHLD    18) SIGCONT    19) SIGSTOP    20) SIGSTP
21) SIGTTIN    22) SIGTTOU    23) SIGURG     24) SIGXCPU    25) SIGXFSZ
26) SIGVTALRM  27) SIGPROF    28) SIGWINCH   29) SIGIO      30) SIGPWR
31) SIGSYS     34) SIGRTMIN   35) SIGRTMIN+1 36) SIGRTMIN+2 37) SIGRTMIN+3
38) SIGRTMIN+4 39) SIGRTMIN+5 40) SIGRTMIN+6 41) SIGRTMIN+7 42) SIGRTMIN+8
43) SIGRTMIN+9 44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+13
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-12
53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7
58) SIGRTMAX-6 59) SIGRTMAX-5 60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2
63) SIGRTMAX-1 64) SIGRTMAX

```

Figure 8. Kill Signals

By default, the kill command sends the **SIGTERM(15)** signal. This signal instructs the process to terminate gracefully. When the user presses Ctrl+C, the **SIGINT (2)** signal is sent to the process. When the user presses Ctrl+Z, the **SIGSTOP (19)** signal is sent.

In the following example, two Terminal program windows are laid out side-by-side. See Figure 9 and Figure 10. Follow these steps:

1. Type **ping 127.0.0.1** in one of the windows.
2. In the other windows, type **ps -el | grep ping** to get the PID of the ping process. (You may get a different PID. Use the one you got.)
3. Type **kill -SIGSTOP** followed by the ping PID and press Enter. The ping process stops. (Note that this command can also be executed as **kill -STOP** or **kill -19**.)
4. In the same window, type **kill -SIGCONT** followed by the ping PID and press Enter. The ping process resumes. (Note that this command can also be executed as **kill -CONT** or **kill -18**.)
5. In the same window, type **kill -SIGKILL** followed by the ping PID and press Enter. The ping process terminates. (Note that this command can also be executed as **kill -KILL** or **kill -9**. Under normal circumstances, the SIGKILL signal should only be used as the last resort. Use the default SIGTERM whenever possible.)
6. Click the ping windows and press Enter to confirm the process is killed.



```

(04/24 22:22:21) cital71@cital71-vm: ~
$ ping 127.0.0.1
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.025 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.031 ms
64 bytes from 127.0.0.1: icmp_seq=3 ttl=64 time=0.084 ms
64 bytes from 127.0.0.1: icmp_seq=4 ttl=64 time=0.090 ms
64 bytes from 127.0.0.1: icmp_seq=5 ttl=64 time=0.082 ms
64 bytes from 127.0.0.1: icmp_seq=6 ttl=64 time=0.082 ms
64 bytes from 127.0.0.1: icmp_seq=7 ttl=64 time=0.082 ms
64 bytes from 127.0.0.1: icmp_seq=8 ttl=64 time=0.082 ms

[1]+  Stopped                  ping 127.0.0.1
(04/24 22:23:20) cital71@cital71-vm: ~
$

(04/24 22:22:13) cital71@cital71-vm: ~
$ ps -el | grep ping
0 S 1000 29556 29548 0 80 0 - 2425 - pts/1
00:00:00 ping
(04/24 22:23:13) cital71@cital71-vm: ~
$ kill -SIGSTOP 29556
(04/24 22:23:20) cital71@cital71-vm: ~
$

```

Figure 9. The SIGSTOP Signal

```

64 bytes from 127.0.0.1: icmp_seq=6 ttl=64 time=0.082 ms
64 bytes from 127.0.0.1: icmp_seq=7 ttl=64 time=0.082 ms
64 bytes from 127.0.0.1: icmp_seq=8 ttl=64 time=0.082 ms

[1]+  Stopped                  ping 127.0.0.1
(04/24 22:23:20) cital71@cital71-vm: ~
$ 64 bytes from 127.0.0.1: icmp_seq=9 ttl=64 time=0.085 ms
64 bytes from 127.0.0.1: icmp_seq=10 ttl=64 time=0.096 ms
64 bytes from 127.0.0.1: icmp_seq=11 ttl=64 time=0.083 ms
64 bytes from 127.0.0.1: icmp_seq=12 ttl=64 time=0.082 ms
64 bytes from 127.0.0.1: icmp_seq=13 ttl=64 time=0.096 ms
64 bytes from 127.0.0.1: icmp_seq=14 ttl=64 time=0.090 ms
64 bytes from 127.0.0.1: icmp_seq=15 ttl=64 time=0.075 ms
64 bytes from 127.0.0.1: icmp_seq=16 ttl=64 time=0.048 ms
64 bytes from 127.0.0.1: icmp_seq=17 ttl=64 time=0.081 ms
64 bytes from 127.0.0.1: icmp_seq=18 ttl=64 time=0.076 ms
64 bytes from 127.0.0.1: icmp_seq=19 ttl=64 time=0.083 ms
64 bytes from 127.0.0.1: icmp_seq=20 ttl=64 time=0.083 ms
64 bytes from 127.0.0.1: icmp_seq=21 ttl=64 time=0.154 ms
64 bytes from 127.0.0.1: icmp_seq=22 ttl=64 time=0.095 ms

(04/24 22:22:13) cital71@cital71-vm: ~
$ ps -el | grep ping
0 S 1000 29556 29548 0 80 0 - 2425 - pts/1
00:00:00 ping
(04/24 22:23:13) cital71@cital71-vm: ~
$ kill -SIGSTOP 29556
(04/24 22:23:20) cital71@cital71-vm: ~
$ kill -SIGCONT 29556
(04/24 22:24:51) cital71@cital71-vm: ~
$ kill -SIGKILL 29556
(04/24 22:25:05) cital71@cital71-vm: ~
$

```

Figure 10. SIGCONT and SIGKILL Signals