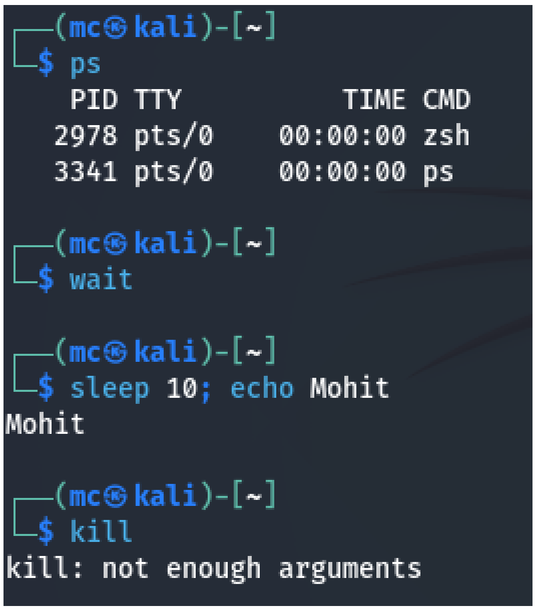
VP Vidyalankar Polytechnic	DEPARTMENT OF COMPUTER ENGINEERING

Subject: Operating System	Subject Code:22516
Semester:5th Semester	Course: Computer Engineering
Laboratory No: L001	Name of Subject Teacher: Prof. Vijay Patil
Name of Student: Siddharth P. Shah	Roll Id: 22203A0041

Experiment No:	5
Title of Experiment	Execute process commands

Practical Set

1. Execute process commands: ps, wait, sleep, exit, kill.



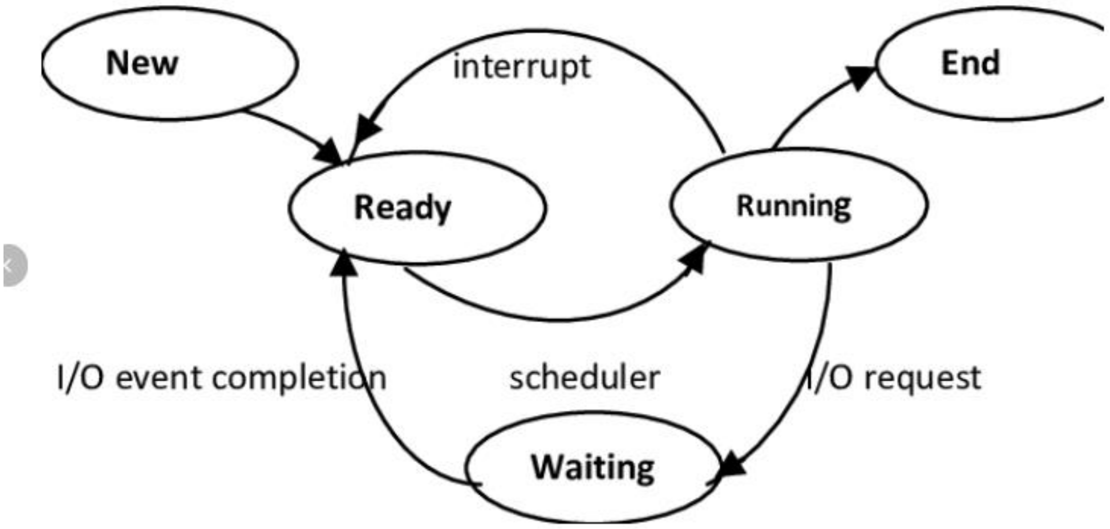
```
┌──(mc㉿kali)-[~]
└─$ ps
    PID TTY          TIME CMD
   2978 pts/0    00:00:00 zsh
   3341 pts/0    00:00:00 ps

┌──(mc㉿kali)-[~]
└─$ wait

┌──(mc㉿kali)-[~]
└─$ sleep 10; echo Mohit
Mohit

┌──(mc㉿kali)-[~]
└─$ kill
kill: not enough arguments
```

2. Explain Process state diagram.



• New State: A process that has just been created but has not yet been admitted to the pool of execution processes by the operating system. Every new operation which is requested to the system is known as the new born process.

• Ready State: When the process is ready to execute but it is waiting for the CPU to execute then this is called as the ready state. After the completion of the input and outputs the process will be on ready state means the process will wait for the processor to execute.

• Running State: The process that is currently being executed. When the process is running under the CPU, or when the program is executed by the CPU, then this is called as the running state process and when a process is running then this will also provide us some outputs on the screen.

• Waiting or Blocked: A process that cannot execute until some event occurs or an I/O completion. When a process is waiting for some input and output operations then this is called as the waiting state. And in this state process is not under the execution instead the process is stored out of memory and when the user will provide the input then this will again be on ready state.

• Terminated State: After the completion of the process, the process will be automatically terminated by the CPU, so this is also called as the terminated state of the process. After executing the whole process, the processor will also de-allocate the memory which is allocated to the process. So this is called as the terminated process.

3. Difference between Process and Thread.

Process	Thread
Process means any program is in execution.	Thread means a segment of a process.
The process takes more time to terminate.	The thread takes less time to terminate.
It takes more time for creation.	It takes less time for creation.
It also takes more time for context switching.	It takes less time for context switching.
The process is less efficient in terms of communication.	Thread is more efficient in terms of communication.
Multiprogramming holds the concepts of multi-process.	We don't need multi programs in action for multiple threads because a single process consists of multiple threads.
The process is isolated.	Threads share memory.
The process is called the heavyweight process.	A Thread is lightweight as each thread in a process shares code, data, and resources.

4. List System Calls for Process Management

- **wait()** - Waits for a child process to terminate and retrieves its exit status.
- **exit()** - Terminates the calling process.
- **getpid()** - Retrieves the process ID of the calling process.
- **kill()** - Sends a signal to a process, often used to terminate it.
- **getppid()** - Retrieves the parent process ID of the calling process.
- **nice()** - Sets the scheduling priority of a process.

5. What is process ID of your login shell?

You can find the process ID (PID) of your login shell by running `echo $$` in your terminal.

6. Give PID of all processes, how will you terminate the processes running on your terminal?

To terminate processes running in your terminal, you can use `kill` followed by the PID of each process.
To get a list of PIDs, you can use `ps` or `pgrep`.
For example, `kill $(pgrep -o -u $USER)` will terminate the oldest process you own.

7. What is difference between wait and sleep?

Wait	**Sleep**
Waits for processes to complete before continuing.	Delays execution for a defined time period.
Can wait for specific process IDs or all background jobs.	Always waits for the exact time specified.
Useful for synchronizing tasks in scripts.	Used to introduce time delays, regardless of process states.

<u>Practical Related Questions:</u>

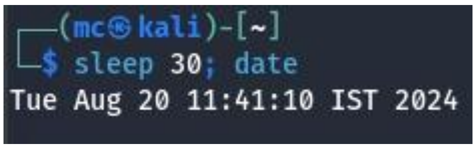
1. **What is name of your login shell?**
   Echo $shell
2. **What are various options of kill command?**
   - **kill 0**: Kills all the processes on the terminal except the login shell by using the special argument 0.
   - **kill 120 230 234**: Kills three processes with PIDs 120, 230, and 234.
   - **kill -9 0**: Kills all processes including the login shell.
   - **kill -9 $$**: Kills the login shell.
3. **What are various options of ps command?**
   - **ps -f**: Full listing showing PPID of each process.
   - **ps -u username**: Displays processes of user username.
   - **ps -a**: Processes of all users.
   - **ps -e**: Processes including user and system processes.
4. **Explain about exit command.**

   Used to quit the shell.

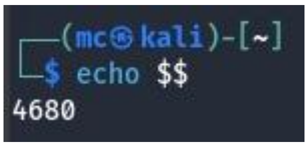
5. **List the system calls for process management.**
   - **wait()** - Waits for a child process to terminate and retrieves its exit status.
   - **exit()** - Terminates the calling process.
   - **getpid()** - Retrieves the process ID of the calling process.
   - **kill()** - Sends a signal to a process, often used to terminate it.
   - **getppid()** - Retrieves the parent process ID of the calling process.
   - **nice()** - Sets the scheduling priority of a process.

XIII. Exercise

1. **Observe the output of the following commands:**
   $sleep 30; date
   $$echo $$

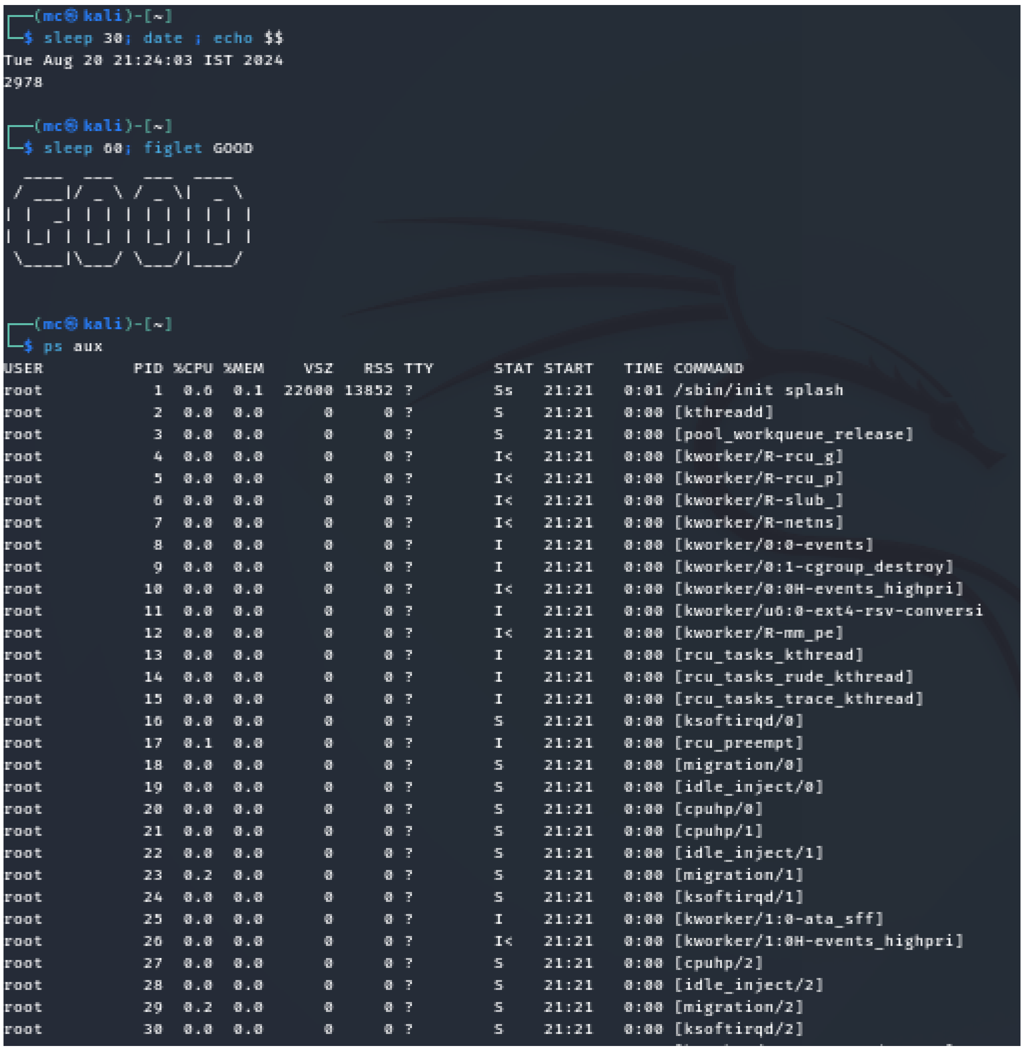


```
┌──(mc㉿kali)-[~]
└─$ sleep 30; date
Tue Aug 20 11:41:10 IST 2024
```



```
┌──(mc㉿kali)-[~]
└─$ echo $$
4680
```

2. **Display full listing of all the processes running on your terminal.**



```
┌──(mc㉿kali)-[~]
└─$ sleep 30; date ; echo $$
Tue Aug 20 21:24:03 IST 2024
2978

┌──(mc㉿kali)-[~]
└─$ sleep 60; figlet GOOD
  ____  ___   ___  ____
 / ___|/ _ \ / _ \|  _ \
| |  _| | | | | | | | | |
| |_| | |_| | |_| | |_| |
 \____|\___/ \___/|____/


┌──(mc㉿kali)-[~]
└─$ ps aux
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root           1  0.6  0.1  22000 13852 ?        Ss   21:21   0:01 /sbin/init splash
root           2  0.0  0.0      0     0 ?        S    21:21   0:00 [kthreadd]
root           3  0.0  0.0      0     0 ?        S    21:21   0:00 [pool_workqueue_release]
root           4  0.0  0.0      0     0 ?        I<   21:21   0:00 [kworker/R-rcu_g]
root           5  0.0  0.0      0     0 ?        I<   21:21   0:00 [kworker/R-rcu_p]
root           6  0.0  0.0      0     0 ?        I<   21:21   0:00 [kworker/R-slub_]
root           7  0.0  0.0      0     0 ?        I<   21:21   0:00 [kworker/R-netns]
root           8  0.0  0.0      0     0 ?        I    21:21   0:00 [kworker/0:0-events]
root           9  0.0  0.0      0     0 ?        I    21:21   0:00 [kworker/0:1-cgroup_destroy]
root          10  0.0  0.0      0     0 ?        I<   21:21   0:00 [kworker/0:0H-events_highpri]
root          11  0.0  0.0      0     0 ?        I    21:21   0:00 [kworker/u6:0-ext4-rsv-conversi
root          12  0.0  0.0      0     0 ?        I<   21:21   0:00 [kworker/R-mm_pe]
root          13  0.0  0.0      0     0 ?        I    21:21   0:00 [rcu_tasks_kthread]
root          14  0.0  0.0      0     0 ?        I    21:21   0:00 [rcu_tasks_rude_kthread]
root          15  0.0  0.0      0     0 ?        I    21:21   0:00 [rcu_tasks_trace_kthread]
root          16  0.0  0.0      0     0 ?        S    21:21   0:00 [ksoftirqd/0]
root          17  0.1  0.0      0     0 ?        I    21:21   0:00 [rcu_preempt]
root          18  0.0  0.0      0     0 ?        S    21:21   0:00 [migration/0]
root          19  0.0  0.0      0     0 ?        S    21:21   0:00 [idle_inject/0]
root          20  0.0  0.0      0     0 ?        S    21:21   0:00 [cpuhp/0]
root          21  0.0  0.0      0     0 ?        S    21:21   0:00 [cpuhp/1]
root          22  0.0  0.0      0     0 ?        S    21:21   0:00 [idle_inject/1]
root          23  0.2  0.0      0     0 ?        S    21:21   0:00 [migration/1]
root          24  0.0  0.0      0     0 ?        S    21:21   0:00 [ksoftirqd/1]
root          25  0.0  0.0      0     0 ?        I    21:21   0:00 [kworker/1:0-ata_sff]
root          26  0.0  0.0      0     0 ?        I<   21:21   0:00 [kworker/1:0H-events_highpri]
root          27  0.0  0.0      0     0 ?        S    21:21   0:00 [cpuhp/2]
root          28  0.0  0.0      0     0 ?        S    21:21   0:00 [idle_inject/2]
root          29  0.2  0.0      0     0 ?        S    21:21   0:00 [migration/2]
root          30  0.0  0.0      0     0 ?        S    21:21   0:00 [ksoftirqd/2]
```

3. **Write output of following commands.**
   $sleep 60; banner GOOD

   ```
   (mc@kali)-[~] $ sleep 60; figlet GOOD

   GOOD
   ```

4. **Write all the process commands.**
   - **ps**: Displays current processes.
   - **top**: Shows a dynamic view of system processes.
   - **htop**: An enhanced, interactive process viewer.
   - **kill**: Sends signals to processes, typically to terminate them.
   - **pkill**: Sends signals to processes based on name.
   - **killall**: Terminates processes by name.
   - **nice**: Starts a process with a modified priority.
   - **renice**: Changes the priority of an already running process.