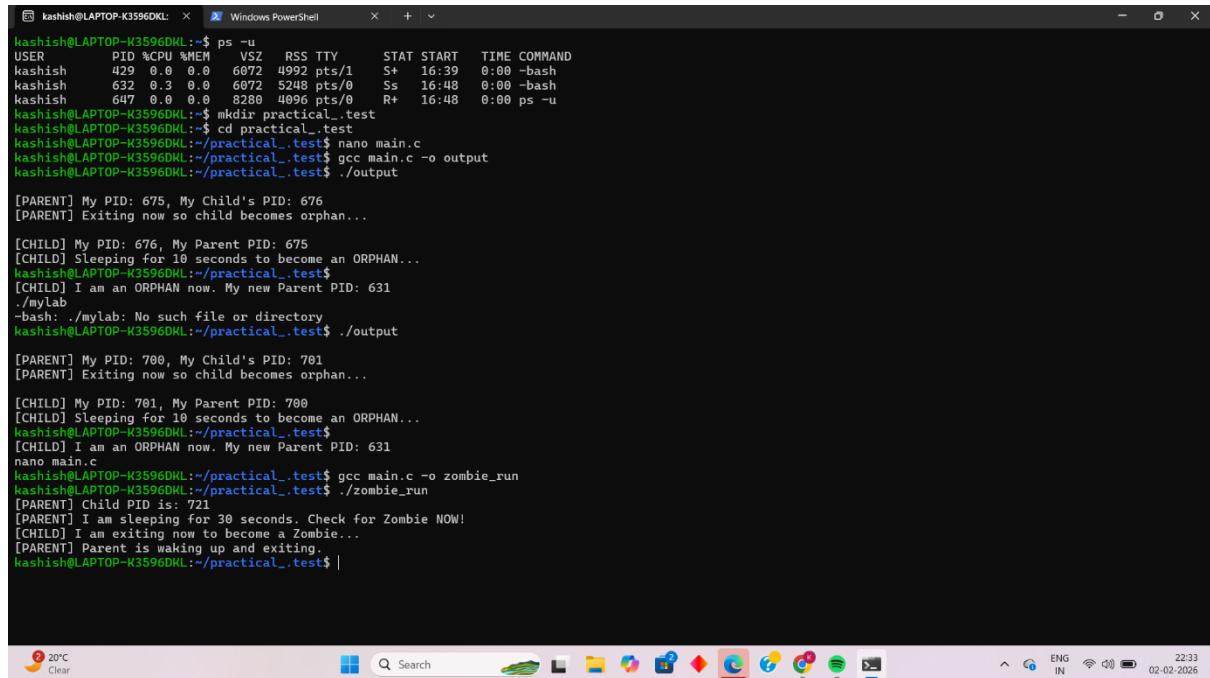


1. Adam is working in an IT company. He has been given a task to reduce the load of a system by killing some of the processes running in the LINUX operating system. Which commands will he use to complete the given task with the help of the following operation?

- Kill processes by name
- Kill a process based on the process name
- Kill a single process at a time with the given process ID



```

kashish@LAPTOP-K3596DKL:~$ ps -a
USER PID %CPU %MEM VSZ RSS TTY STAT START TIME COMMAND
kashish 429 0.0 0.0 6072 4992 pts/1 S+ 16:39 0:00 -bash
kashish 632 0.3 0.0 6072 5248 pts/0 Ss 16:48 0:00 -bash
kashish 647 0.0 0.0 8280 4096 pts/0 R+ 16:48 0:00 ps -u
kashish@LAPTOP-K3596DKL:~$ mkdir practical_test
kashish@LAPTOP-K3596DKL:~/practical_test$ nano main.c
kashish@LAPTOP-K3596DKL:~/practical_test$ gcc main.c -o output
kashish@LAPTOP-K3596DKL:~/practical_test$ ./output

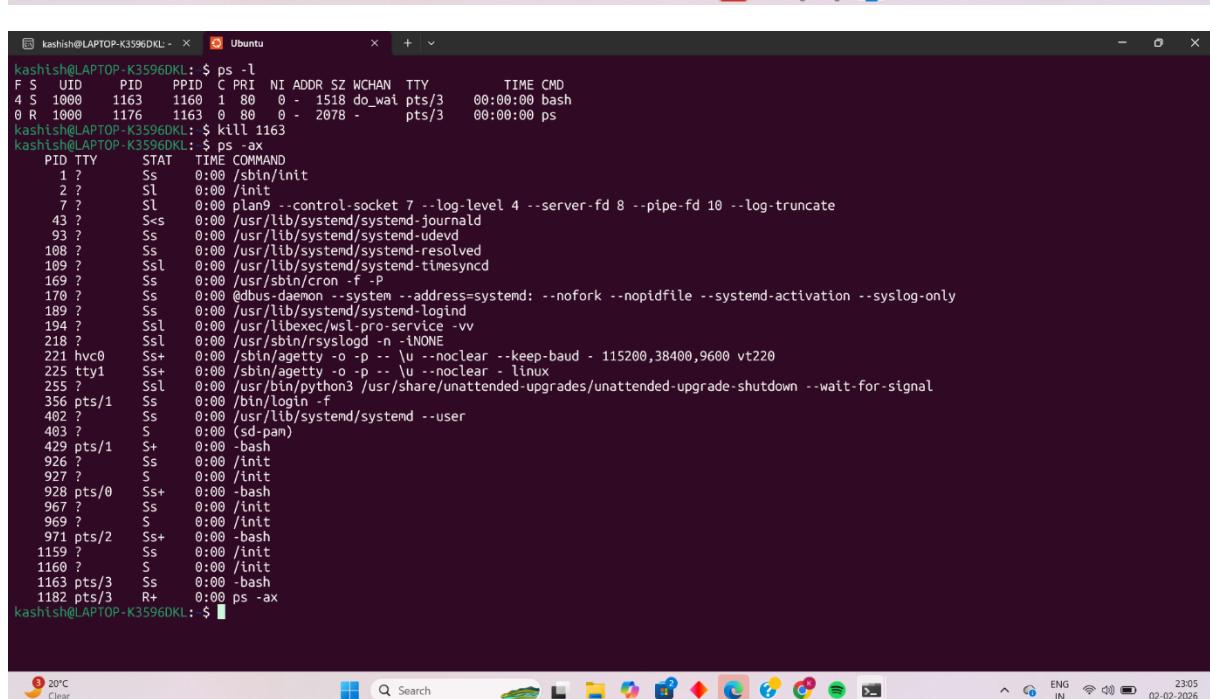
[PARENT] My PID: 675, My Child's PID: 676
[PARENT] Exiting now so child becomes orphan...

[CHILD] My PID: 676, My Parent PID: 675
[CHILD] Sleeping for 10 seconds to become an ORPHAN...
kashish@LAPTOP-K3596DKL:~/practical_test$ 
[CHILD] I am an ORPHAN now. My new Parent PID: 631
./mylab
:bash: ./mylab: No such file or directory
kashish@LAPTOP-K3596DKL:~/practical_test$ ./output

[PARENT] My PID: 700, My Child's PID: 701
[PARENT] Exiting now so child becomes orphan...

[CHILD] My PID: 701, My Parent PID: 700
[CHILD] Sleeping for 10 seconds to become an ORPHAN...
kashish@LAPTOP-K3596DKL:~/practical_test$ 
[CHILD] I am an ORPHAN now. My new Parent PID: 631
nano main.c
kashish@LAPTOP-K3596DKL:~/practical_test$ gcc main.c -o zombie_run
kashish@LAPTOP-K3596DKL:~/practical_test$ ./zombie_run
[PARENT] Child PID is: 721
[PARENT] I am sleeping for 30 seconds. Check for Zombie NOW!
[CHILD] I am exiting now to become a Zombie...
[PARENT] Parent is waking up and exiting.
kashish@LAPTOP-K3596DKL:~/practical_test$ 

```



```

kashish@LAPTOP-K3596DKL:~$ ps -l
F S UID PID PPID C PRI NI ADDR SZ WCHAN TTY TIME CMD
4 S 1000 1163 1160 1 80 0 - 1518 do_wai pts/3 00:00:00 bash
0 R 1000 1176 1163 0 80 0 - 2078 - pts/3 00:00:00 ps
kashish@LAPTOP-K3596DKL:~$ kill 1163
kashish@LAPTOP-K3596DKL:~$ ps -ax
PID TTY STAT TIME COMMAND
1 ? Ss 0:00 /sbin/init
2 ? Sl 0:00 /init
7 ? Sl 0:00 plan9 --control-socket 7 --log-level 4 --server-fd 8 --pipe-fd 10 --log-truncate
43 ? Ss< 0:00 /usr/lib/systemd/systemd-journald
93 ? Ss 0:00 /usr/lib/systemd/systemd-udevd
108 ? Ss 0:00 /usr/lib/systemd/systemd-resolved
109 ? Ssl 0:00 /usr/lib/systemd/systemd-timesyncd
169 ? Ss 0:00 /usr/sbin/cron -f -P
170 ? Ss 0:00 @ibus-daemon --system --address=systemd: --nofork --nopidfile --systemd-activation --syslog-only
189 ? Ss 0:00 /usr/lib/systemd/systemd-logind
194 ? Ssl 0:00 /usr/libexec/wsl-pro-service -vv
218 ? Ssl 0:00 /usr/sbin/syslogd -n -INONE
221 hvc0 Ss+ 0:00 /sbin/agetty -o -p -- \u --noclear --keep-baud - 115200,38400,9600 vt220
225 tty1 Ss+ 0:00 /sbin/agetty -o -p -- \u --noclear - linux
255 ? Ssl 0:00 /usr/bin/python3 /usr/share/unattended-upgrades/unattended-upgrade-shutdown --wait-for-signal
356 pts/1 Ss 0:00 /bin/login -f
402 ? Ss 0:00 /usr/lib/systemd/systemd --user
403 ? S 0:00 (sd-pam)
429 pts/1 S+ 0:00 -bash
926 ? Ss 0:00 /init
927 ? S 0:00 /init
928 pts/0 Ss+ 0:00 -bash
967 ? Ss 0:00 /init
969 ? S 0:00 /init
971 pts/2 Ss+ 0:00 -bash
1159 ? Ss 0:00 /init
1160 ? S 0:00 /init
1163 pts/3 Ss 0:00 -bash
1182 pts/3 R+ 0:00 ps -ax
kashish@LAPTOP-K3596DKL:~$ 

```

## 2. Write a program for process creation using C

- Orphan Process

- Zombie Process

```
kashish@LAPTOP-K3596DKL:~$ nano zombie_code.c
kashish@LAPTOP-K3596DKL:~$ gcc zombie_code.c -o zombie_exe
kashish@LAPTOP-K3596DKL:~$ ./zombie_exe
[PARENT] My Child is PID 1006. I am sleeping for 100s.
[PARENT] GO TO THE OTHER TAB NOW AND TYPE 'ps -l'
[CHILD] I am dead now. My PID was: 1006
kashish@LAPTOP-K3596DKL:~$ |
```

The screenshot shows a Windows desktop environment with a terminal window titled "Ubuntu". The terminal is displaying the output of a C program named "zombie\_code.c". The program creates a child process and sleeps for 100 seconds. The parent process then prints the child's PID and exits. The child process continues to run, but its status is listed as "dead" in the terminal output.

```
GNU nano 7.2                                         zombie_code.c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>

int main() {
    pid_t pid = fork();

    if (pid == 0) {
        // Child process finishes immediately
        printf("[CHILD] I am dead now. My PID was: %d\n", getpid());
        exit(0);
    }
    else {
        // Parent process sleeps for 100 seconds
        printf("[PARENT] My Child is PID %d. I am sleeping for 100s.\n", pid);
        printf("[PARENT] GO TO THE OTHER TAB NOW AND TYPE 'ps -l'\n";
        sleep(100);
        exit(0);
    }
}
```

The screenshot shows a Windows desktop environment with a terminal window titled "Ubuntu". The terminal is displaying the source code of a C program named "zombie\_code.c". The code uses the fork() function to create a child process. The parent process then sleeps for 100 seconds. The child process exits immediately, while the parent continues to run.

```
kashish@LAPTOP-K3596DKL: ~
```

```
108 ? Ss+ 0:00 /usr/lib/systemd/systemd-resolved
109 ? Ss+ 0:00 /usr/lib/systemd/systemd-timesyncd
109 ? Ss+ 0:00 @dbus-daemon --system --address=systemd: --nofork --nopidfile --systemd-activation --syslog-only
109 ? Ss+ 0:00 /usr/lib/systemd/systemd-logind
109 ? Ss+ 0:00 /usr/lib/systemd/systemd-journal-service --no-pid --no-vv
218 ? Ssl 0:00 /usr/sbin/syslogd -n -LNONE
221 hvc0 Ss+ 0:00 /sbin/getty -o p -- u -noclear -keep-baud - 115200,38400,9600 vt22
225 ttv1 Ss+ 0:00 /sbin/getty -o p -- u -noclear -linux
227 pts/0 Ssl 0:00 /bin/python3 /usr/share/unattended-upgrades/unattended-upgrade-shutdown --wait-for-signal
356 pts/1 Ss+ 0:00 /bin/login
492 ? Ss+ 0:00 /usr/lib/systemd/systemd -user
493 ? S+ 0:00 sdd-pam
494 pts/1 S+ 0:00 bash
926 ? S+ 0:00 /init
927 ? S+ 0:00 /init
928 pts/0 Ss+ 0:00 bash
930 ? S+ 0:00 /init
939 ? S+ 0:00 /init
971 pts/2 Ss+ 0:00 bash
1159 ? Ss+ 0:00 /init
1163 pts/3 Ss+ 0:00 bash
1182 pts/3 R+ 0:00 ps -ax
kashish@LAPTOP-K3596DKL: $ ps -u
USER PID %CPU %MEM VSZ RSS TTY STAT START TIME COMMAND
kashish 597 0.0 0.0 6072 4992 pts/1 S+ 16:39 0:00 bash
kashish 928 0.0 0.0 6072 5248 pts/0 S+ 17:16 0:00 bash
kashish 971 0.0 0.0 6072 5248 pts/2 S+ 17:17 0:00 bash
kashish 1163 0.0 0.0 6072 5248 pts/3 S+ 17:34 0:00 bash
kashish 1202 0.0 0.0 6072 5248 pts/3 R+ 17:38 0:00 ps -u
kashish@LAPTOP-K3596DKL: $ pskill zombie_xve
kashish@LAPTOP-K3596DKL: $ ps -u
USER PID %CPU %MEM VSZ RSS TTY STAT START TIME COMMAND
kashish 597 0.0 0.0 6072 4992 pts/1 S+ 16:39 0:00 bash
kashish 928 0.0 0.0 6072 5248 pts/0 S+ 17:16 0:00 bash
kashish 971 0.0 0.0 6072 5248 pts/2 S+ 17:17 0:00 bash
kashish 1163 0.0 0.0 6072 5248 pts/3 S+ 17:34 0:00 bash
kashish 1211 0.0 0.0 8288 4996 pts/3 R+ 17:40 0:00 ps -u
kashish@LAPTOP-K3596DKL: $
```

### 3. Create the process using fork () system call.

- Child Process creation
- Parent process creation
- PPID and PID

```
kashish@LAPTOP-K3596DKL: ~
```

```
kashish@LAPTOP-K3596DKL: ~$ nano task3.c
kashish@LAPTOP-K3596DKL: ~$ gcc task3.c -o task3_output
kashish@LAPTOP-K3596DKL: ~$ ./task3_output
```

```
---- PARENT PROCESS ---
My PID (Process ID): 1725
My Child's PID: 1726

---- CHILD PROCESS ---
My PID (Process ID): 1726
My Parent's PID (PPID): 1725
kashish@LAPTOP-K3596DKL: ~$ |
```

```
GNU nano 7.2                                     task3.c
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>

int main() {
    pid_t p;
    p = fork(); // Creating the child process

    if (p < 0) {
        printf("Fork Failed\n");
    }
    else if (p == 0) {
        // This block is executed by the CHILD
        printf("\n--- CHILD PROCESS ---");
        printf("\nMy PID (Process ID): %d", getpid());
        printf("\nMy Parent's PID (PPID): %d\n", getppid());
    }
    else {
        // This block is executed by the PARENT
        printf("\n--- PARENT PROCESS ---");
        printf("\nMy PID (Process ID): %d", getpid());
        printf("\nMy Child's PID: %d\n", p);
    }

    return 0;
}
```

The screenshot shows a terminal window titled "task3.c" containing the provided C code. Below the terminal is a standard Linux desktop interface with a menu bar, a dock with various application icons, and a system tray at the bottom right.