Course name: OPERATING SYSTEMS LAB

Course code: DSE 3161

No of contact hours/week: 3

Credit: 1

By
Dr. Sandhya Parasnath Dubey
Assistant Professor,
Department of Data Science and Computer Applications - MIT, Manipal
Academy of Higher Education, Manipal-576104, Karnataka, India.
Mob: +91-9886542135
Email: sandhya.dubey@manipal.edu

"Our greatest weakness lies in giving up. The most certain way to succeed is always to try just one more time"

PROCESS SYSTEM CALLS

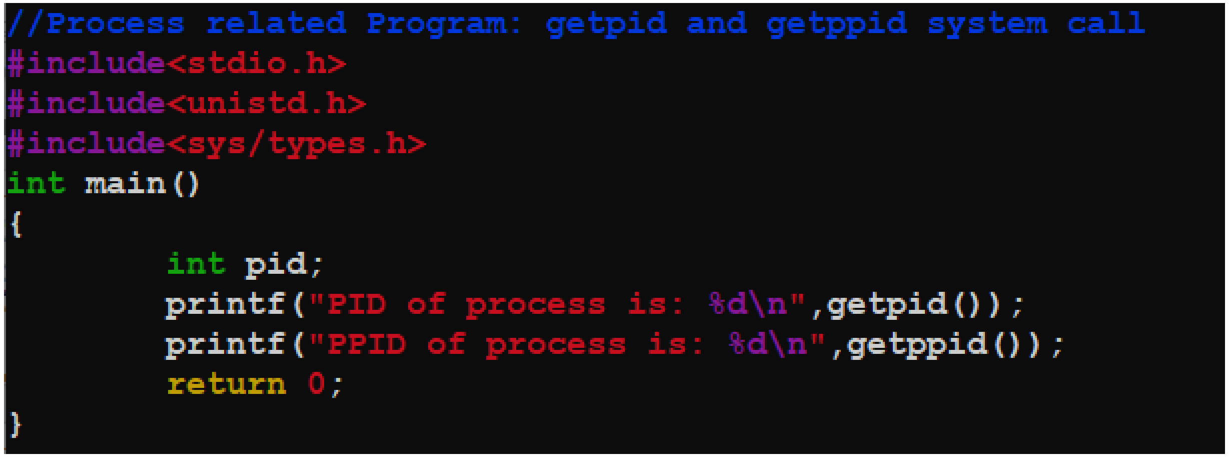
System Calls Related to Processes

getpid()

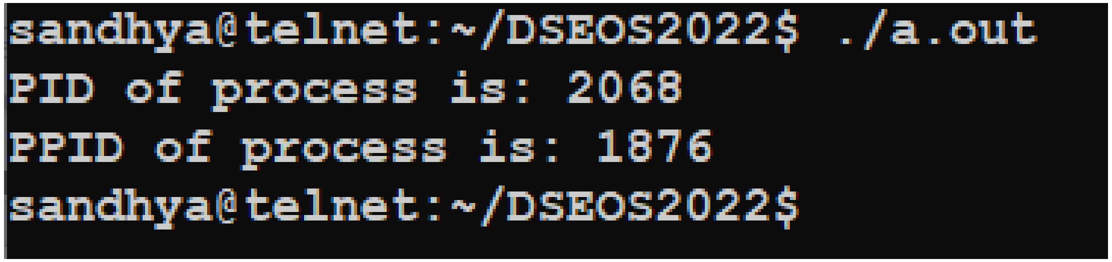
This function returns the **process identifiers** of the calling process.

```
#include <sys/types.h>
#include <unistd.h>
pid_t getpid(void);    // this function returns the process identifier (PID)
pid_t getppid(void); // this function returns the parent process identifier (PPID)
```

getpid() and getppid()



```
//Process related Program: getpid and getppid system call
#include<stdio.h>
#include<unistd.h>
#include<sys/types.h>
int main()
{
        int pid;
        printf("PID of process is: %d\n",getpid());
        printf("PPID of process is: %d\n",getppid());
        return 0;
}
```



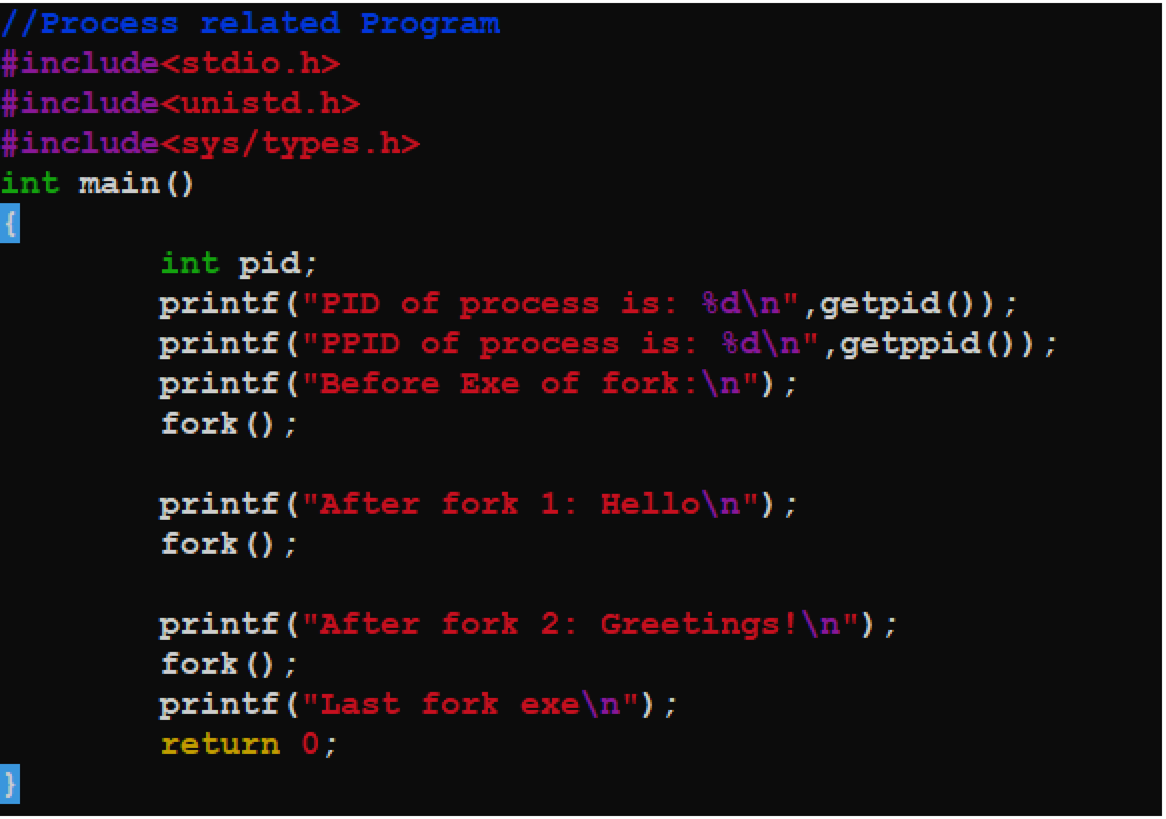
```
sandhya@telnet:~/DSEOS2022$ ./a.out
PID of process is: 2068
PPID of process is: 1876
sandhya@telnet:~/DSEOS2022$
```

fork()

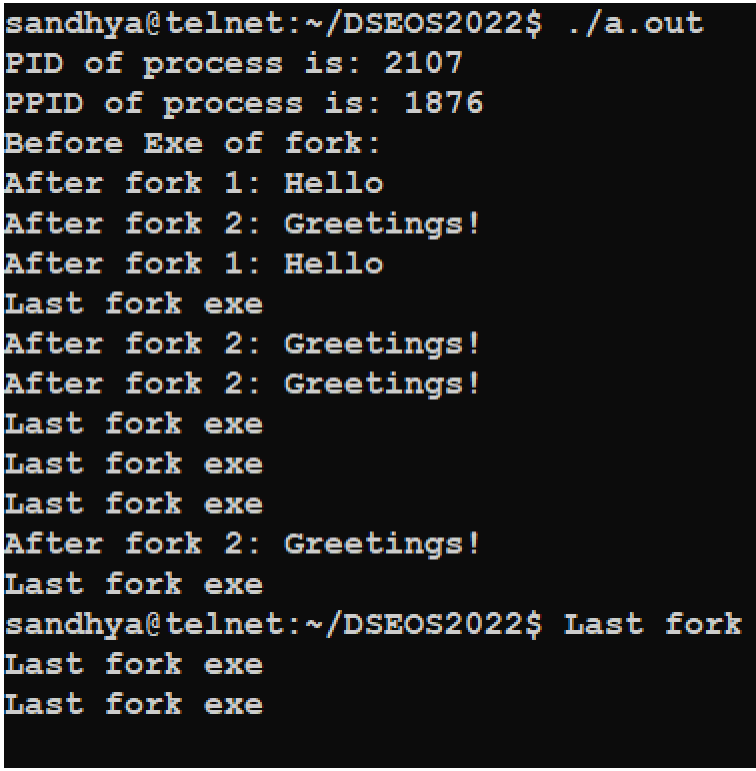
- A new process is created by calling fork.
- This system call duplicates the current process, creating a new entry in the process table with many of the same attributes as the current process.
- The new process is almost identical to the original, executing the same code but with its own data space, environment, and file descriptors.
- After a new child process is created, both processes will execute the next instruction following the fork() system call.

```
#include <sys/types.h>
#include <unistd.h>
pid_t fork(void);
```

fork()

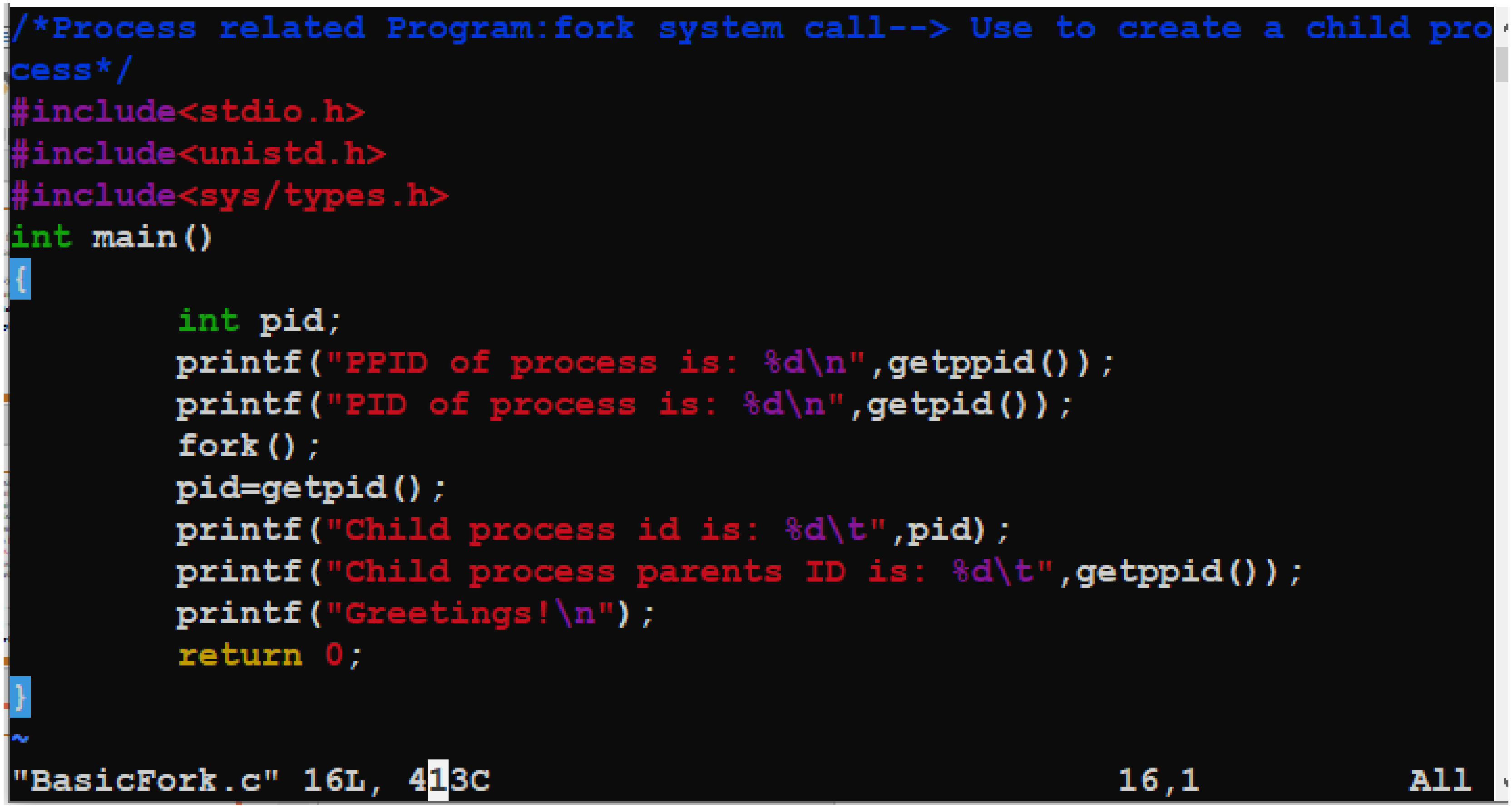


```
//Process related Program
#include<stdio.h>
#include<unistd.h>
#include<sys/types.h>
int main()
{
        int pid;
        printf("PID of process is: %d\n",getpid());
        printf("PPID of process is: %d\n",getppid());
        printf("Before Exe of fork:\n");
        fork();

        printf("After fork 1: Hello\n");
        fork();

        printf("After fork 2: Greetings!\n");
        fork();
        printf("Last fork exe\n");
        return 0;
}
```

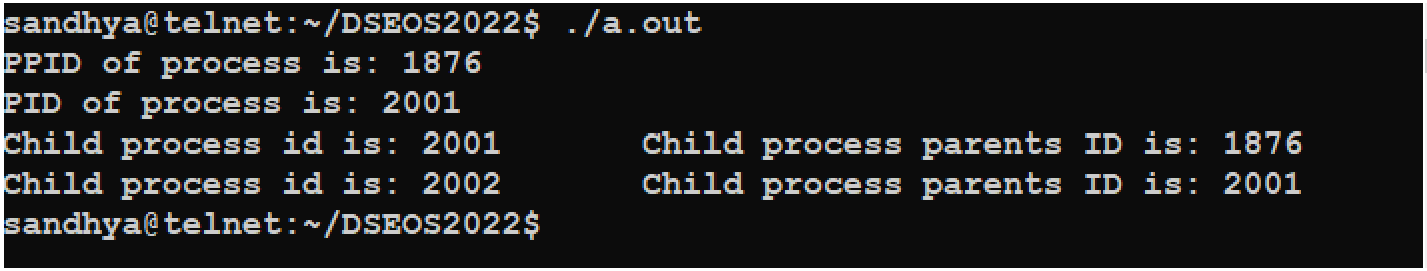


```
sandhya@telnet:~/DSEOS2022$ ./a.out
PID of process is: 2107
PPID of process is: 1876
Before Exe of fork:
After fork 1: Hello
After fork 2: Greetings!
After fork 1: Hello
Last fork exe
After fork 2: Greetings!
After fork 2: Greetings!
Last fork exe
Last fork exe
Last fork exe
After fork 2: Greetings!
Last fork exe
sandhya@telnet:~/DSEOS2022$ Last fork
Last fork exe
Last fork exe
```

fork()



```
/*Process related Program:fork system call--> Use to create a child pro
cess*/
#include<stdio.h>
#include<unistd.h>
#include<sys/types.h>
int main()
{
        int pid;
        printf("PPID of process is: %d\n",getppid());
        printf("PID of process is: %d\n",getpid());
        fork();
        pid=getpid();
        printf("Child process id is: %d\t",pid);
        printf("Child process parents ID is: %d\t",getppid());
        printf("Greetings!\n");
        return 0;
}
~
"BasicFork.c" 16L, 413C                                  16,1          All
```



```
sandhya@telnet:~/DSEOS2022$ ./a.out
PPID of process is: 1876
PID of process is: 2001
Child process id is: 2001       Child process parents ID is: 1876
Child process id is: 2002       Child process parents ID is: 2001
sandhya@telnet:~/DSEOS2022$
```

Linux Sleep Command (Pause a Bash Script)

- Command-line utility that allows you to suspends the calling process for a specified time

- Pauses the execution of the next command for a given number of seconds.

```
sleep NUMBER[SUFFIX]
```

- The NUMBER may be a positive integer or a floating-point number.
- The SUFFIX may be one of the following:

  s - seconds (default)
  m - minutes
  h - hours
  d - days

wait

- built-in command of Linux that waits for completing any running process
- used with a particular process id or job id
- If no process id or job id is given with wait command then it will wait for all current child processes to complete and returns exit status

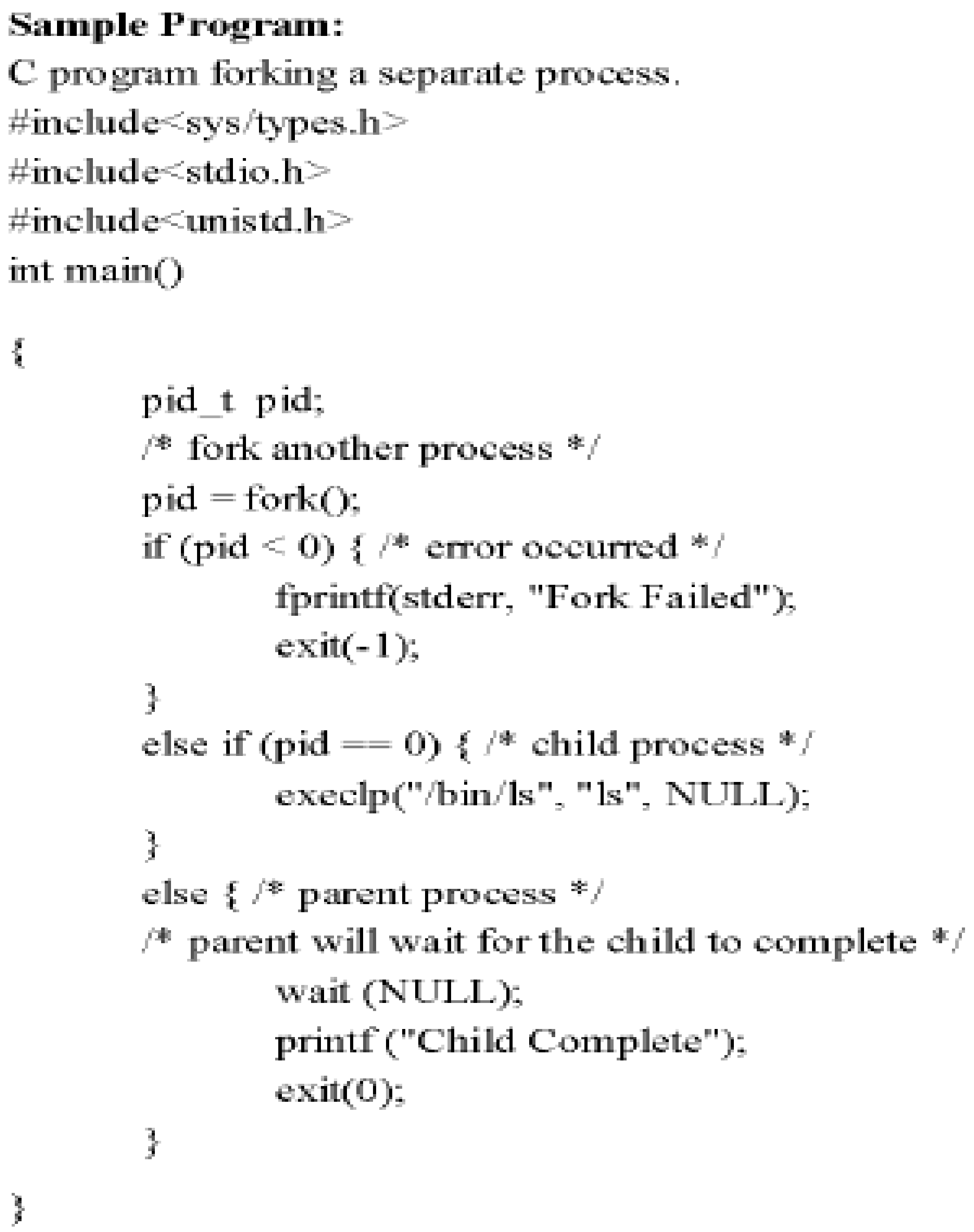
```
#!/bin/bash
echo "testing wait command1" &
process_id=$!
echo "testing wait command2" &
wait $process_id
echo Job 1 exited with status $?
wait $!
echo Job 2 exited with status $?
```

Exit() System Call

- This system call is used to terminate the current running process.
- A value of zero is passed to indicate that the execution of process was successful.
- A non-zero value is passed if the execution of process was unsuccessful.
- All shell commands are written in C including grep. grep will return 0 through exit if the command is successfully runs (grep could find pattern in file). If grep fails to find pattern in file, then it will call exit() with a non-zero value.
- This is applicable to all commands.

The exec() System Call

- ✓ The exec function will execute a specified program passed as argument to it, in the same process.
- ✓ The exec() will not create a new process. As new process is not created, the process ID (PID) does not change across an execute, but the data and code of the calling process are replaced by those of the new process.
- ✓ fork() is the name of the system call that the parent process uses to "divide" itself ("fork") into two identical processes.
- ✓ After calling fork(), the created child process is actually an exact copy of the parent, which would probably be of limited use, so it replaces itself with another process using the system call exec().



Sample Program:

C program forking a separate process.

```
#include<sys/types.h>
#include<stdio.h>
#include<unistd.h>
int main()

{
        pid_t  pid;
        /* fork another process */
        pid = fork();
        if (pid < 0) { /* error occurred */
                fprintf(stderr, "Fork Failed");
                exit(-1);
        }
        else if (pid == 0) { /* child process */
                execlp("/bin/ls", "ls", NULL);
        }
        else { /* parent process */
        /* parent will wait for the child to complete */
                wait (NULL);
                printf ("Child Complete");
                exit(0);
        }

}
```

The versions of exec are

- execl
- execle
- Execv
- execve
- execlp
- execvp

The naming convention: exec*

'l' indicates a list arrangement (a series of null terminated arguments)

'v' indicate the array or vector arrangement (like the argv structure).

'e' Indicates the programmer will construct (in the array/vector format) and pass their own environment variable list

'p' indicates the current PATH string should be used when the system searches for executable files.

- ✓ The parent process can either continue execution or watt for the child process to complete.
- ✓ If the parent chooses to wait for the child to die, then the parent will
  receive the exit code of the program that the child executed.
- ✓ If a parent does not wait for the child, and the child terminates before the parent, then the child is called zombie process.
- ✓ If a parent terminates before the child process then the child is attached to a process called init (whose PID is 1).
- ✓ In this case, whenever the child does not have a parent then child is called orphan process.

Exercise

1. Write a C program to block a parent process until child completes using wait system call.

2. Write a program to create a child process. Display the process IDs of the process, parent and child (if any) in both the parent and child processes.

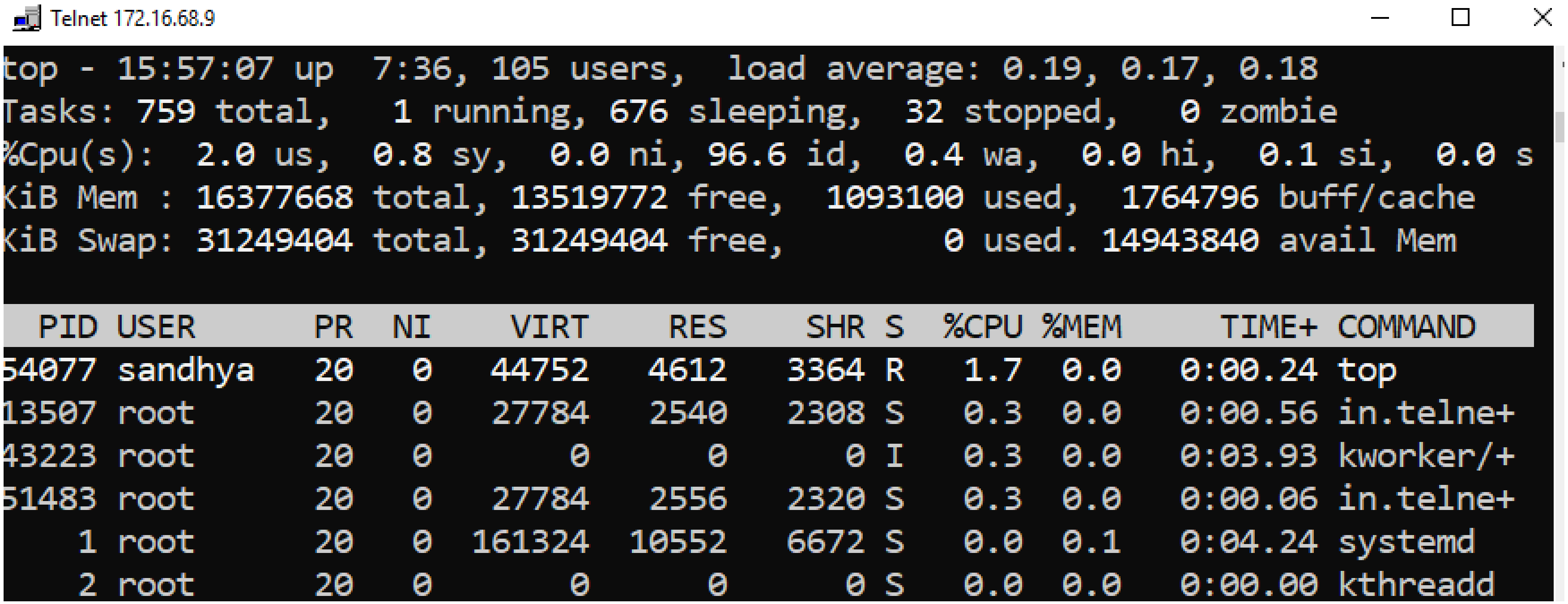
3. Accept an array of integers. Display the unsorted array in the parent process. Create a child process. Sort and display the sorted array in the child process.

4. Create a orphan process (parent dies before child, child process adopted by "init" process) and display the PID of parent of child before and after it becomes orphan. Use sleep (n) in the child to delay the termination.

5. Write a C program to simulate ls command

Top

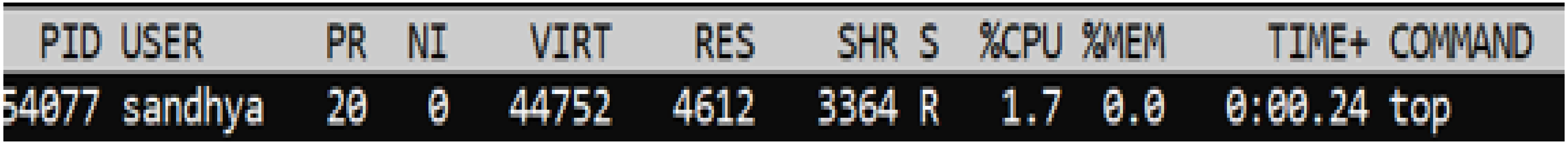
This utility tells the user about all the running processes on the Linux machine.



```
top - 15:57:07 up  7:36, 105 users,  load average: 0.19, 0.17, 0.18
Tasks: 759 total,   1 running, 676 sleeping,  32 stopped,   0 zombie
%Cpu(s):  2.0 us,  0.8 sy,  0.0 ni, 96.6 id,  0.4 wa,  0.0 hi,  0.1 si,  0.0 s
KiB Mem : 16377668 total, 13519772 free,  1093100 used,  1764796 buff/cache
KiB Swap: 31249404 total, 31249404 free,        0 used. 14943840 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
54077 sandhya   20   0   44752   4612   3364 R   1.7  0.0   0:00.24 top
13507 root      20   0   27784   2540   2308 S   0.3  0.0   0:00.56 in.telne+
43223 root      20   0       0      0      0 I   0.3  0.0   0:03.93 kworker/+
51483 root      20   0   27784   2556   2320 S   0.3  0.0   0:00.06 in.telne+
    1 root      20   0  161324  10552   6672 S   0.0  0.1   0:04.24 systemd
    2 root      20   0       0      0      0 S   0.0  0.0   0:00.00 kthreadd
```

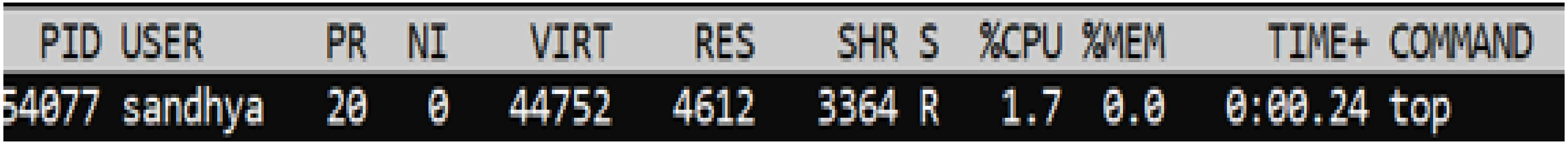
Press 'q' on the keyboard to move out of the process display

Field	Description	Example 1	Example 2
PID	The process ID of each task	1525	961
User	The username of task owner	Home	Root
PR	Priority Can be 20(highest) or -20(lowest)	20	20
NI	The nice value of a task	0	0
VIRT	Virtual memory used (kb)	1775	75972
RES	Physical memory used (kb)	100	51
SHR	Shared memory used (kb)	28	7952



```
  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
54077 sandhya   20   0   44752   4612   3364 R   1.7  0.0   0:00.24 top
```

S	Status There are five types: 'D' = uninterruptible sleep 'R' = running 'S' = sleeping 'T' = traced or stopped 'Z' = zombie	S	R
%CPU	% of CPU time	1.7	1.0
%MEM	Physical memory used	10	5.1
TIME+	Total CPU time	5:05.34	2:23.42
Command	Command name	Photoshop.exe	Xorg

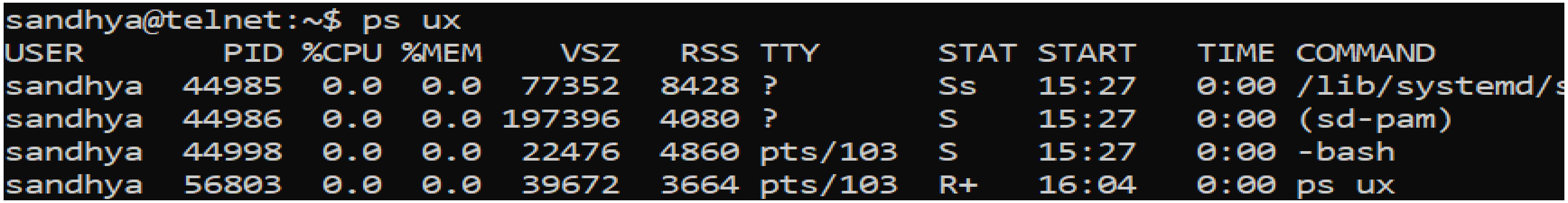


```
  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
54077 sandhya   20   0   44752   4612   3364 R   1.7  0.0   0:00.24 top
```

Tracking ongoing processes

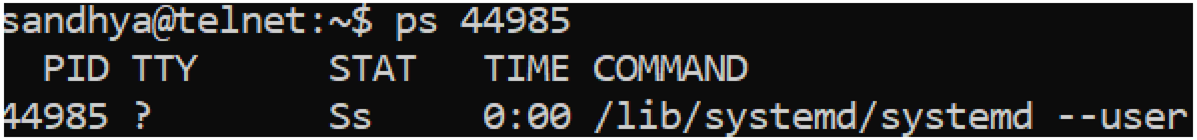
PS

- This command stands for 'Process Status'. It is similar to the "Task Manager" that pop-ups in a Windows Machine when we use Cntrl+Alt+Del.
- This command is similar to 'top' command but the information displayed is different.
- To check all the processes running under a user, use the command ps ux



```
sandhya@telnet:~$ ps ux
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
sandhya    44985  0.0  0.0  77352  8428 ?        Ss   15:27   0:00 /lib/systemd/s
sandhya    44986  0.0  0.0 197396  4080 ?        S    15:27   0:00 (sd-pam)
sandhya    44998  0.0  0.0  22476  4860 pts/103  S    15:27   0:00 -bash
sandhya    56803  0.0  0.0  39672  3664 pts/103  R+   16:04   0:00 ps ux
```

- For a single process information, ps along with process id is used ps PID



```
sandhya@telnet:~$ ps 44985
  PID TTY      STAT   TIME COMMAND
44985 ?        Ss     0:00 /lib/systemd/systemd --user
```

Stopping a process

Kill

- **terminates running processes** on a Linux machine.

  kill PID

- To find the PID of a process

  pidof Process name

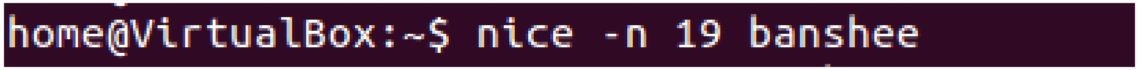
- When running in foreground, hitting Ctrl + c (interrupt character) will exit the command

- If a process ignores a regular kill command, you can use kill -9 followed by the process ID

NICE

- Linux can run a lot of processes at a time, which can slow down the speed of some high priority processes and result in poor performance.
- To avoid this, you can tell your machine to prioritize processes as per your requirements.
- This priority is called Niceness in Linux, and it has a value between -20 to 19. The lower the Niceness index, the higher would be a priority given to that task.
- The default value of all the processes is 0.
- To start a process with a niceness value other than the default value use the following syntax

  nice -n 'Nice value' process name

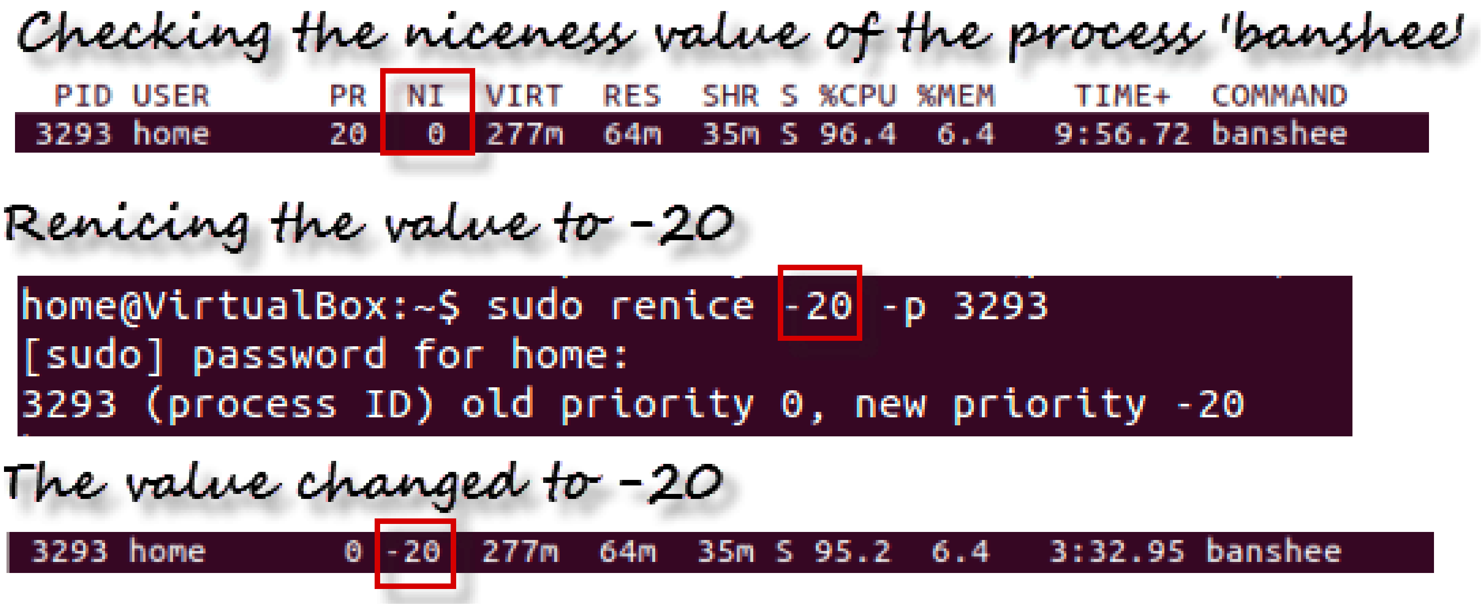


```
home@VirtualBox:~$ nice -n 19 banshee
```

- If there is some process already running on the system, then you can 'Renice' its value using syntax.

  renice 'nice value' -p 'PID'

- To change Niceness, you can use the 'top' command to determine the PID (process id) and its Nice value. Later use the renice command to change the value.



Checking the niceness value of the process 'banshee'

```
  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 3293 home      20   0  277m  64m  35m S 96.4  6.4   9:56.72 banshee
```

Renicing the value to -20

```
home@VirtualBox:~$ sudo renice -20 -p 3293
[sudo] password for home:
3293 (process ID) old priority 0, new priority -20
```

The value changed to -20

```
 3293 home       0 -20  277m  64m  35m S 95.2  6.4   3:32.95 banshee
```

Command	Description
bg	To send a process to the background
fg	To run a stopped process in the foreground
top	Details on all Active Processes
ps	Give the status of processes running for a user
ps PID	Gives the status of a particular process
pidof	Gives the Process ID (PID) of a process
kill PID	Kills a process
nice	Starts a process with a given priority
renice	Changes priority of an already running process
df	Gives free hard disk space on your system
free	Gives free RAM on your system

Summary:

- Any running program or a command given to a Linux system is called a process
- A process could run in foreground or background
- The priority index of a process is called Nice in Linux. Its default value is 0, and it can vary between 20 to -19
- The lower the Niceness index, the higher would be priority given to that task