

OPERATING SYSTEMS

LAB DIGITAL ASSIGNMENT - 2

Course Code : SWE3001

Slot : L25+L26

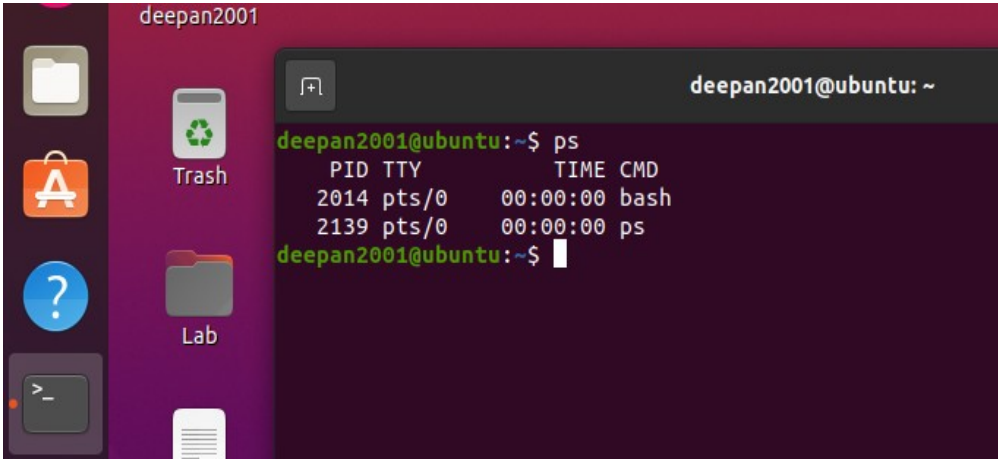
Name : S.Deepan

Reg No : 19MIS0102

Monitoring Processes :

1. List All Processes in Current Shell

If you run ps command without any arguments, it displays processes for the current shell.



The screenshot shows a terminal window titled 'deepan2001@ubuntu: ~'. The user has entered the command 'ps'. The output is as follows:

```
deepan2001@ubuntu:~$ ps
  PID TTY          TIME CMD
 2014 pts/0    00:00:00 bash
 2139 pts/0    00:00:00 ps
```

The terminal window also shows a sidebar with icons for 'Trash' and 'Lab'.

2. Print All Processes in Different Formats

Display every active process on a Linux system in generic (Unix/Linux) format.

```

deepan2001@ubuntu:~$ ps -A
  PID TTY          TIME CMD
    1 ?            00:00:03 systemd
    2 ?            00:00:00 kthreadd
    3 ?            00:00:00 rcu_gp
    4 ?            00:00:00 rcu_par_gp
    6 ?            00:00:00 kworker/0:0H-events_highpri
    9 ?            00:00:00 mm_percpu_wq
   10 ?            00:00:00 rcu_tasks_rude_
   11 ?            00:00:00 rcu_tasks_trace
   12 ?            00:00:00 ksoftirqd/0
   13 ?            00:00:00 rcu_sched
   14 ?            00:00:00 migration/0
   15 ?            00:00:00 idle_inject/0
   16 ?            00:00:00 cpuhp/0
   17 ?            00:00:00 cpuhp/1
   18 ?            00:00:00 idle_inject/1
   19 ?            00:00:00 migration/1
   20 ?            00:00:00 ksoftirqd/1
   22 ?            00:00:00 kworker/1:0H-events_highpri
   23 ?            00:00:00 kdevtmpfs
   24 ?            00:00:00 netns
   25 ?            00:00:00 inet_frag_wq
   26 ?            00:00:00 kauditd
   28 ?            00:00:00 khungtaskd
   29 ?            00:00:00 oom_reaper
   30 ?            00:00:00 writeback
   31 ?            00:00:00 kcompactd0

```

```

deepan2001@ubuntu:~$ ps -e
  PID TTY          TIME CMD
    1 ?            00:00:03 systemd
    2 ?            00:00:00 kthreadd
    3 ?            00:00:00 rcu_gp
    4 ?            00:00:00 rcu_par_gp
    6 ?            00:00:00 kworker/0:0H-events_highpri
    9 ?            00:00:00 mm_percpu_wq
   10 ?            00:00:00 rcu_tasks_rude_
   11 ?            00:00:00 rcu_tasks_trace
   12 ?            00:00:00 ksoftirqd/0
   13 ?            00:00:00 rcu_sched
   14 ?            00:00:00 migration/0
   15 ?            00:00:00 idle_inject/0
   16 ?            00:00:00 cpuhp/0
   17 ?            00:00:00 cpuhp/1
   18 ?            00:00:00 idle_inject/1
   19 ?            00:00:00 migration/1
   20 ?            00:00:00 ksoftirqd/1
   22 ?            00:00:00 kworker/1:0H-events_highpri
   23 ?            00:00:00 kdevtmpfs
   24 ?            00:00:00 netns
   25 ?            00:00:00 inet_frag_wq
   26 ?            00:00:00 kauditd
   28 ?            00:00:00 khungtaskd
   29 ?            00:00:00 oom_reaper
   30 ?            00:00:00 writeback
   31 ?            00:00:00 kcompactd0

```

```

deepan2001@ubuntu:~$ ps -ef
UID          PID    PPID  C  STIME TTY          TIME CMD
root           1        0  0  01:34 ?           00:00:03 /sbin/init auto noprompt
root           2        0  0  01:34 ?           00:00:00 [kthreadd]
root           3        2  0  01:34 ?           00:00:00 [rcu_gp]
root           4        2  0  01:34 ?           00:00:00 [rcu_par_gp]
root           6        2  0  01:34 ?           00:00:00 [kworker/0:0H-events_highpri]
root           9        2  0  01:34 ?           00:00:00 [mm_percpu_wq]
root          10        2  0  01:34 ?           00:00:00 [rcu_tasks_rude_]
root          11        2  0  01:34 ?           00:00:00 [rcu_tasks_trace]
root          12        2  0  01:34 ?           00:00:00 [ksoftirqd/0]
root          13        2  0  01:34 ?           00:00:00 [rcu_sched]
root          14        2  0  01:34 ?           00:00:00 [migration/0]
root          15        2  0  01:34 ?           00:00:00 [idle_inject/0]
root          16        2  0  01:34 ?           00:00:00 [cpuhp/0]
root          17        2  0  01:34 ?           00:00:00 [cpuhp/1]
root          18        2  0  01:34 ?           00:00:00 [idle_inject/1]
root          19        2  0  01:34 ?           00:00:00 [migration/1]
root          20        2  0  01:34 ?           00:00:00 [ksoftirqd/1]
root          22        2  0  01:34 ?           00:00:00 [kworker/1:0H-events_highpri]
root          23        2  0  01:34 ?           00:00:00 [kdevtmpfs]
root          24        2  0  01:34 ?           00:00:00 [netns]
root          25        2  0  01:34 ?           00:00:00 [inet_frag_wq]
root          26        2  0  01:34 ?           00:00:00 [kauditd]
root          28        2  0  01:34 ?           00:00:00 [khungtaskd]
root          29        2  0  01:34 ?           00:00:00 [oom_reaper]
root          30        2  0  01:34 ?           00:00:00 [writeback]
root          31        2  0  01:34 ?           00:00:00 [kcompactd0]
root          32        2  0  01:34 ?           00:00:00 [ksmd]

```

3. Display all processes in BSD format.

```

deepan2001@ubuntu:~$ ps au
USER          PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
deepan2+     1447  0.0   0.3 172652  6396 tty2    Ssl+  01:34   0:00 /usr/lib/gdm3
deepan2+     1450  0.3   3.4 295700 68976 tty2    Sl+   01:34   0:14 /usr/lib/xorg
deepan2+     1505  0.0   0.7 199288 15380 tty2    Sl+   01:34   0:00 /usr/libexec/
deepan2+     2023  0.0   0.2  19248  4880 pts/0    Ss    01:36   0:00 bash
deepan2+     2746  0.0   0.1  20132  3252 pts/0    R+    02:41   0:00 ps au
deepan2001@ubuntu:~$

```

4. Display User Running Processes

You can select all processes owned by you using below command:

```

deepan2001@ubuntu:~$ ps -ax
  PID TTY          STAT TIME COMMAND
    1 ?           Ss    0:03 /sbin/init auto noprompt
    2 ?           S      0:00 [kthreadd]
    3 ?           I<     0:00 [rcu_gp]
    4 ?           I<     0:00 [rcu_par_gp]
    5 ?           I      0:00 [kworker/0:0-events]
    6 ?           I<     0:00 [kworker/0:0H-events_highpri]
    8 ?           I      0:01 [kworker/u256:0-ext4-rsv-conversion]
    9 ?           I<     0:00 [mm_percpu_wq]
   10 ?          S      0:00 [rcu_tasks_rude_]
   11 ?          S      0:00 [rcu_tasks_trace]
   12 ?          S      0:00 [ksoftirqd/0]
   13 ?          I      0:00 [rcu_sched]
   14 ?          S      0:00 [migration/0]
   15 ?          S      0:00 [idle_inject/0]
   16 ?          S      0:00 [cpuhp/0]
   17 ?          S      0:00 [cpuhp/1]
   18 ?          S      0:00 [idle_inject/1]
   19 ?          S      0:00 [migration/1]
   20 ?          S      0:00 [ksoftirqd/1]
   22 ?          I<     0:00 [kworker/1:0H-events_highpri]
   23 ?          S      0:00 [kdevtmpfs]

```

5. Print All Processes Running as Root

The command below enables you to view every process running with root user privileges

```

deepan2001@ubuntu:~$ ps -U root -u root
  PID TTY          TIME CMD
    1 ?           00:00:03 systemd
    2 ?           00:00:00 kthreadd
    3 ?           00:00:00 rcu_gp
    4 ?           00:00:00 rcu_par_gp
    5 ?           00:00:00 kworker/0:0-events
    6 ?           00:00:00 kworker/0:0H-events_highpri
    9 ?           00:00:00 mm_percpu_wq
   10 ?           00:00:00 rcu_tasks_rude_
   11 ?           00:00:00 rcu_tasks_trace
   12 ?           00:00:00 ksoftirqd/0
   13 ?           00:00:00 rcu_sched
   14 ?           00:00:00 migration/0
   15 ?           00:00:00 idle_inject/0
   16 ?           00:00:00 cpuhp/0
   17 ?           00:00:00 cpuhp/1
   18 ?           00:00:00 idle_inject/1
   19 ?           00:00:00 migration/1
   20 ?           00:00:00 ksoftirqd/1
   22 ?           00:00:00 kworker/1:0H-events_highpri
   23 ?           00:00:00 kdevtmpfs
   24 ?           00:00:00 netns
   25 ?           00:00:00 inet_frag_wq
   26 ?           00:00:00 kauditd
   27 ?           00:00:00 kworker/0:2-events
   28 ?           00:00:00 khungtaskd
   29 ?           00:00:00 oom_reaper
   30 ?           00:00:00 writeback

```

6. Print Process Tree

A process tree shows how processes on the system are linked to each other; processes whose parents have been killed are adopted by the init (or systemd).

```
deepan2001@ubuntu:~$ ps -e --forest
PID TTY          TIME CMD
  2 ?            00:00:00 kthreadd
  3 ?            00:00:00  \_ rcu_gp
  4 ?            00:00:00  \_ rcu_par_gp
  5 ?            00:00:00  \_ kworker/0:0-events
  6 ?            00:00:00  \_ kworker/0:0H-events_highpri
  9 ?            00:00:00  \_ mm_percpu_wq
 10 ?            00:00:00  \_ rcu_tasks_rude_
 11 ?            00:00:00  \_ rcu_tasks_trace
 12 ?            00:00:00  \_ ksoftirqd/0
 13 ?            00:00:00  \_ rcu_sched
 14 ?            00:00:00  \_ migration/0
 15 ?            00:00:00  \_ idle inject/0
```

7. Print Process Threads

To print all threads of a process, use the -C flag, this will show the LWP (light weight process) as well as NLWP (number of light weight process) columns.

```
deepan2001@ubuntu:~$ ps -elf
UID      PID  PPID  LWP  C  NLWP  STIME  TTY          TIME CMD
root      1    0     1  0    1  07:53  ?           00:00:03 /sbin/init auto noprompt
root      2    0     2  0    1  07:53  ?           00:00:00 [kthreadd]
root      3    2     3  0    1  07:53  ?           00:00:00 [rcu_gp]
root      4    2     4  0    1  07:53  ?           00:00:00 [rcu_par_gp]
root      5    2     5  0    1  07:53  ?           00:00:00 [kworker/0:0-events]
root      6    2     6  0    1  07:53  ?           00:00:00 [kworker/0:0H-events_highpri]
root      9    2     9  0    1  07:53  ?           00:00:00 [mm_percpu_wq]
root     10    2    10  0    1  07:53  ?           00:00:00 [rcu_tasks_rude_]
root     11    2    11  0    1  07:53  ?           00:00:00 [rcu_tasks_trace]
root     12    2    12  0    1  07:53  ?           00:00:00 [ksoftirqd/0]
root     13    2    13  0    1  07:53  ?           00:00:00 [rcu_sched]
root     14    2    14  0    1  07:53  ?           00:00:00 [migration/0]
root     15    2    15  0    1  07:53  ?           00:00:00 [idle inject/0]
```

```

deepan2001@ubuntu:~$ ps axms

```

UID	PID	PENDING	BLOCKED	IGNORED	CAUGHT	STAT	TTY	TIME	COMMAND
0	1	0000000000000000	-	-	-	-	?	0:03	/sbin/init auto noprompt
0	-	0000000000000000 7be3c0fe28014a03	0000000000001000	00000001800004ec	Ss	-	-	0:03	-
0	2	0000000000000000	-	-	-	-	?	0:00	[kthreadd]
0	-	0000000000000000	0000000000000000	ffffffffffffffff	0000000000000000	S	-	0:00	-
0	3	0000000000000000	-	-	-	-	?	0:00	[rcu_gp]
0	-	0000000000000000	0000000000000000	ffffffffffffffff	0000000000000000	I<	-	0:00	-
0	4	0000000000000000	-	-	-	-	?	0:00	[rcu_par_gp]
0	-	0000000000000000	0000000000000000	ffffffffffffffff	0000000000000000	I<	-	0:00	-
0	5	0000000000000000	-	-	-	-	?	0:00	[kworker/0:0-events]
0	-	0000000000000000	0000000000000000	ffffffffffffffff	0000000000000000	I	-	0:00	-
0	6	0000000000000000	-	-	-	-	?	0:00	[kworker/0:0H-events_highpri]
0	-	0000000000000000	0000000000000000	ffffffffffffffff	0000000000000000	I<	-	0:00	-
0	9	0000000000000000	-	-	-	-	?	0:00	[mm_percpu_wq]
0	-	0000000000000000	0000000000000000	ffffffffffffffff	0000000000000000	I<	-	0:00	-
0	10	0000000000000000	-	-	-	-	?	0:00	[rcu_tasks_rude_]
0	-	0000000000000000	0000000000000000	ffffffffffffffff	0000000000000000	S	-	0:00	-
0	11	0000000000000000	-	-	-	-	?	0:00	[rcu_tasks_trace]
0	-	0000000000000000	0000000000000000	ffffffffffffffff	0000000000000000	S	-	0:00	-
0	12	0000000000000000	-	-	-	-	?	0:00	[ksoftirqd/0]

```

deepan2001@ubuntu:~$ ps -fL

```

UID	PID	PPID	LWP	C	NLWP	STIME	TTY	TIME	CMD
deepan2+	2014	2006	2014	0	1	07:55	pts/0	00:00:00	bash
deepan2+	2322	2014	2322	0	1	08:09	pts/0	00:00:00	ps -fL

```

deepan2001@ubuntu:~$

```

```

deepan2001@ubuntu:~$ ps -fL -C pager

```

UID	PID	PPID	LWP	C	NLWP	STIME	TTY	TIME	CMD
-----	-----	------	-----	---	------	-------	-----	------	-----

```

deepan2001@ubuntu:~$

```

Creating Multiple Processes :

Program 1 :

```

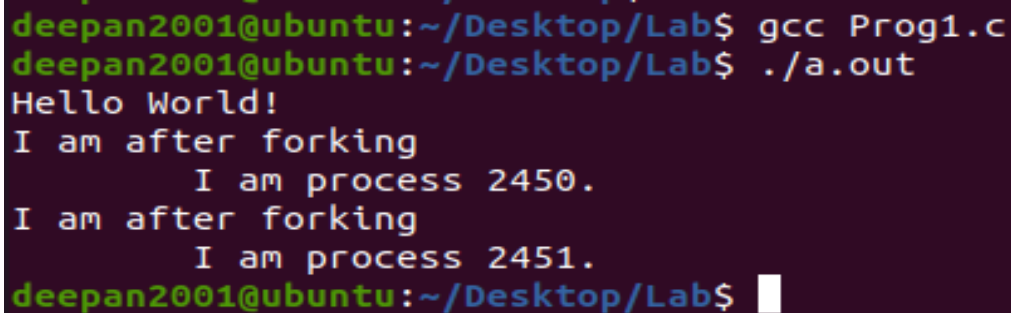
#include <stdio.h>

#include <unistd.h> /* contains fork prototype */

int main(void)
{
    printf("Hello World!\n");
    fork( );
    printf("I am after forking\n");
}

```

```
printf("\tI am process %d.\n", getpid( ));  
}
```



```
deepan2001@ubuntu:~/Desktop/Lab$ gcc Prog1.c  
deepan2001@ubuntu:~/Desktop/Lab$ ./a.out  
Hello World!  
I am after forking  
        I am process 2450.  
I am after forking  
        I am process 2451.  
deepan2001@ubuntu:~/Desktop/Lab$
```

Program 2 :

```
#include <stdio.h>  
#include <unistd.h> /* contains fork prototype */  
int main(void)  
{  
    int pid;  
    printf("Hello World!\n");  
    printf("I am the parent process and pid is : %d .\n",getpid());  
    printf("Here i am before use of forking\n");  
    pid = fork();  
    printf("Here I am just after forking\n");  
    if (pid == 0)  
  
        printf("I am the child process and pid is :%d.\n",getpid());  
    else  
        printf("I am the parent process and pid is: %d .\n",getpid());  
}
```



```

deepan2001@ubuntu:~/Desktop/Lab$ gcc Prog2.c
deepan2001@ubuntu:~/Desktop/Lab$ ./a.out
Hello World!
I am the parent process and pid is : 2543 .
Here i am before use of forking
Here I am just after forking
I am the parent process and pid is: 2543 .
Here I am just after forking
I am the child process and pid is :2544.
deepan2001@ubuntu:~/Desktop/Lab$ █

```

Program 3 :

```

#include <stdio.h>
#include <unistd.h> /* contains fork prototype */
int main(void)
{
printf("Here I am just before first forking statement\n");
fork();
printf("Here I am just after first forking statement\n");
fork();
printf("Here I am just after second forking statement\n");
printf("\t\tHello World from process %d!\n", getpid());
}

```

```

deepan2001@ubuntu:~/Desktop/Lab$ gcc Prog3.c
deepan2001@ubuntu:~/Desktop/Lab$ ./a.out
Here I am just before first forking statement
Here I am just after first forking statement
Here I am just after first forking statement
Here I am just after second forking statement
        Hello World from process 2630!
Here I am just after second forking statement
        Hello World from process 2631!
Here I am just after second forking statement
        Hello World from process 2632!
deepan2001@ubuntu:~/Desktop/Lab$ Here I am just after second forking statement
        Hello World from process 2633!
█

```


Process Completion :

Program 4 :

/*Guarantees the child process will print its message before the parent process*/

```
#include <stdio.h>
```

```
#include <sys/wait.h> /* contains prototype for wait */
```

```
main(void)
```

```
{ int pid;
```

```
int status;
```

```
printf("Hello World!\n");
```

```
pid = fork( );
```

```
if (pid == -1) /* check for error in fork */
```

```
{ perror("bad fork");
```

```
exit(1);
```

```
}
```

```
if (pid == 0)
```

```
printf("I am the child process.\n");
```

```
else
```

```
{
```

```
wait(&status); /* parent waits for child to finish */
```

```
printf("I am the parent process.\n");
```

```
} }
```

```
deepan2001@ubuntu:~/Desktop/Lab$ ./a.out
```

```
Hello World!
```

```
I am the child process.
```

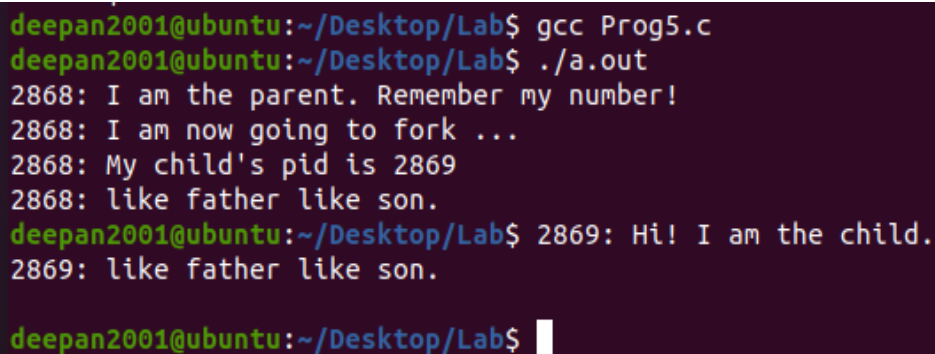
```
I am the parent process.
```

```
deepan2001@ubuntu:~/Desktop/Lab$
```

Program 5 :

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/wait.h>

main()
{
    int forkresult;
    printf("%d: I am the parent. Remember my number!\n", getpid());
    printf("%d: I am now going to fork ... \n", getpid());
    forkresult = fork();
    if (forkresult != 0)
    { /* the parent will execute this code */
        printf("%d: My child's pid is %d\n", getpid(), forkresult);
    }
    else /* forkresult == 0 */
    { /* the child will execute this code */
        printf("%d: Hi! I am the child.\n", getpid());
    }
    printf("%d: like father like son. \n", getpid());
}
```



```
deepan2001@ubuntu:~/Desktop/Lab$ gcc Prog5.c
deepan2001@ubuntu:~/Desktop/Lab$ ./a.out
2868: I am the parent. Remember my number!
2868: I am now going to fork ...
2868: My child's pid is 2869
2868: like father like son.
deepan2001@ubuntu:~/Desktop/Lab$ 2869: Hi! I am the child.
2869: like father like son.
deepan2001@ubuntu:~/Desktop/Lab$
```

Orphan processes :

Program 6 :

```
#include <stdio.h>

main()
{
    int pid ;
    printf("I'am the original process with PID %d and PPID %d.\n",
    getpid(), getppid()) ;
    pid = fork ( ) ; /* Duplicate. Child and parent continue from here */
    if ( pid != 0 ) /* pid is non-zero,so I must be the parent*/
    {
        printf("I'am the parent with PID %d and PPID %d.\n",
        getpid(), getppid()) ;
        printf("My child's PID is %d\n", pid ) ;
    }
    else /* pid is zero, so I must be the child */
    {
        sleep(4); /* make sure that the parent terminates first */
        printf("I'm the child with PID %d and PPID %d.\n",
        getpid(), getppid()) ;
    }
    printf ("PID %d terminates.\n", getpid()) ; }
```

```
deepan2001@ubuntu:~/Desktop/Lab$ ./a.out
I'am the original process with PID 4152 and PPID 2014.
I'am the parent with PID 4152 and PPID 2014.
My child's PID is 4153
PID 4152 terminates.
deepan2001@ubuntu:~/Desktop/Lab$ I'm the child with PID 4153 and PPID 1459.
PID 4153 terminates.
```

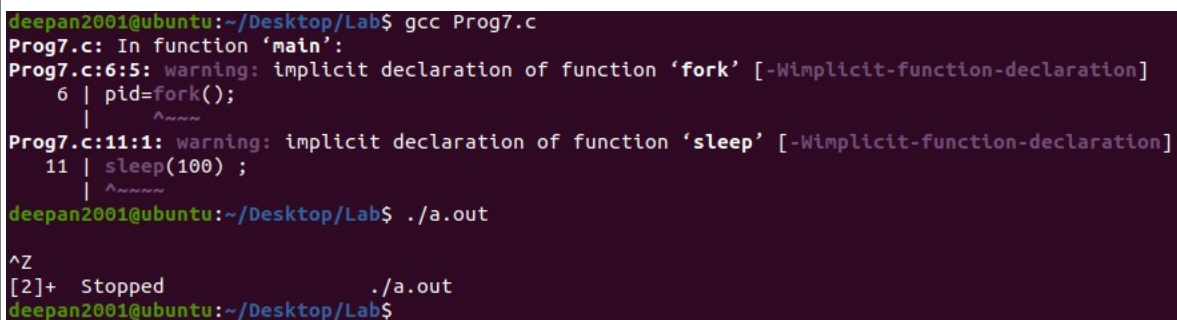
Zombie processes :

Program 7 :

```
#include <stdio.h>

main ( )
{
    int pid ;

    pid = fork(); /* Duplicate. Child and parent continue from here */
    if ( pid != 0 ) /* pid is non-zero, so I must be the parent */
    {
        while (1) /* Never terminate and never execute a wait ( ) */
        sleep (100) ; /* stop executing for 100 seconds */
    }
    else /* pid is zero, so I must be the child */
    {
        exit (42) ; /* exit with any number */
    }
}
```



```
deepan2001@ubuntu:~/Desktop/Lab$ gcc Prog7.c
Prog7.c: In function 'main':
Prog7.c:6:5: warning: implicit declaration of function 'fork' [-Wimplicit-function-declaration]
   6 | pid=fork();
     |           ^
Prog7.c:11:11: warning: implicit declaration of function 'sleep' [-Wimplicit-function-declaration]
   11 | sleep(100) ;
      |           ^
deepan2001@ubuntu:~/Desktop/Lab$ ./a.out
^Z
[2]+  Stopped                  ./a.out
deepan2001@ubuntu:~/Desktop/Lab$
```

Example :

Test.c

```
#include <stdio.h>
#include <unistd.h>
```

```
#include <stdio.h>

int main(int argc, char*argv[])
{
char*args[] = {"Hello", "All",NULL};
printf("About to run a new program\n");
execv("./np",args);
perror("execv");
printf("End of main program");
return 0;
}
```

New.c

```
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
int main(int argc, char *argv[])
{
int i;
for(i=0;i<argc;i++)
printf("%s\n",argv[i]);
return 0;
}
```

Compile & Execute:

```
$ gcc -o Test Test.c
```

```
$ gcc -o New New.c
```

```
$ ./Test
```

```
deepan2001@ubuntu:~/Desktop/Lab$ gcc -o Test Test.c
deepan2001@ubuntu:~/Desktop/Lab$ gcc -o New New.c
deepan2001@ubuntu:~/Desktop/Lab$ ./Test
About to run a new program
execv: No such file or directory
End of main program
deepan2001@ubuntu:~/Desktop/Lab$
deepan2001@ubuntu:~/Desktop/Lab$ ./New
./New
deepan2001@ubuntu:~/Desktop/Lab$
```

ASSESSMENT QUESTIONS :

Sample Program 1 :

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
puts("Before fork");
fork();
puts("After fork");
return 0;
}
```

```
deepan2001@ubuntu:~/Desktop/Lab$ gcc Sample_Prog1.c
deepan2001@ubuntu:~/Desktop/Lab$ ./a.out
Before fork
After fork
After fork
deepan2001@ubuntu:~/Desktop/Lab$
```

Perform the following operations and answer the questions :

1. How many lines are printed by the program?

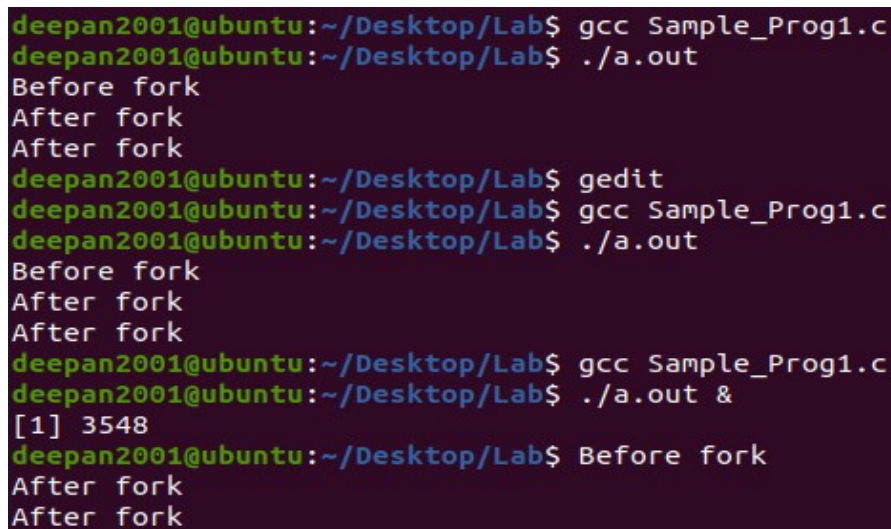
Three lines are printed in the above program they are :

Before fork, After fork, After fork

2. Describe what is happening to produce the answer observed for the above question.

From the above program all the lines are executed by the parent program till the line above fork creation. Parent program starts execution and prints “before fork”. The parent program execution started and first it prints “before fork”. Then after the before fork execution, the execution stops for 10secs and then it resumes and again started its execution of further commands and now it prints the output as “After fork” 2times(because 1st after fork is printed by one child process and 2nd parent prints “after fork”).

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main(){
puts("Before fork");
fork();
sleep(10);
puts("After fork");
return 0;}
```



```
deepan2001@ubuntu:~/Desktop/Lab$ gcc Sample_Prog1.c
deepan2001@ubuntu:~/Desktop/Lab$ ./a.out
Before fork
After fork
After fork
deepan2001@ubuntu:~/Desktop/Lab$ gedit
deepan2001@ubuntu:~/Desktop/Lab$ gcc Sample_Prog1.c
deepan2001@ubuntu:~/Desktop/Lab$ ./a.out
Before fork
After fork
After fork
deepan2001@ubuntu:~/Desktop/Lab$ gcc Sample_Prog1.c
deepan2001@ubuntu:~/Desktop/Lab$ ./a.out &
[1] 3548
deepan2001@ubuntu:~/Desktop/Lab$ Before fork
After fork
After fork
█
```


run Program 1 in the background (use &)

`./a.out &`

3. Consult the man pages for the ps (process status) utility; they will help you determine how to display and interpret the various types of information that is reported. Then, using the appropriate options, observe and report the PIDs and the status (i.e. state info) of your executing program. Provide a brief explanation of your observations.

```
deepan2001@ubuntu:~/Desktop/Lab$ gcc Sample_Prog1.c
deepan2001@ubuntu:~/Desktop/Lab$ ./a.out
Before fork
After fork
After fork
deepan2001@ubuntu:~/Desktop/Lab$ ps
  PID TTY          TIME CMD
 3601 pts/1        00:00:00 bash
 3628 pts/1        00:00:00 ps
deepan2001@ubuntu:~/Desktop/Lab$
```

ps - report a snapshot of the current processes

ps

PID	TTY	TIME	CMD
3601	pts/0	00:00:00	bash
3628	pts/0	00:00:00	ps

when the ps personality setting is BSD-like. The set of processes selected in this manner is in addition to the set of processes selected by other means.

An alternate description is that this option causes ps to list all processes with a terminal (tty), or to list all processes when used together with the x option

```

deepan2001@ubuntu:~/Desktop/Lab$ ps
  PID TTY          TIME CMD
 3601 pts/1    00:00:00 bash
 3628 pts/1    00:00:00 ps
deepan2001@ubuntu:~/Desktop/Lab$ ps a
  PID TTY          STAT TIME  COMMAND
 1505 tty2      Ssl+  0:00  /usr/lib/gdm3/gdm-x-session --run-script env GNOME_
 1511 tty2      Sl+   1:18  /usr/lib/xorg/Xorg vt2 -displayfd 3 -auth /run/user
 1566 tty2      Sl+   0:00  /usr/libexec/gnome-session-binary --systemd --syste
 2014 pts/0      Ss+   0:00  bash
 3601 pts/1      Ss    0:00  bash
 3648 pts/1      R+    0:00  ps a
deepan2001@ubuntu:~/Desktop/Lab$

```

Sample Program 2 :

```

#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
int main(int argc, char* argv[])
{
    int i, limit;
    if (argc < 2){
        fputs("Usage : must supply a limit value\n", stderr);
        exit(1);}
    limit = atoi(argv[1]);
    fork();
    fork();
    printf("PID#: %d\n", getpid());
    for(i=0; i<limit; i++)
        printf("%d\n",i);
    return 0;
}

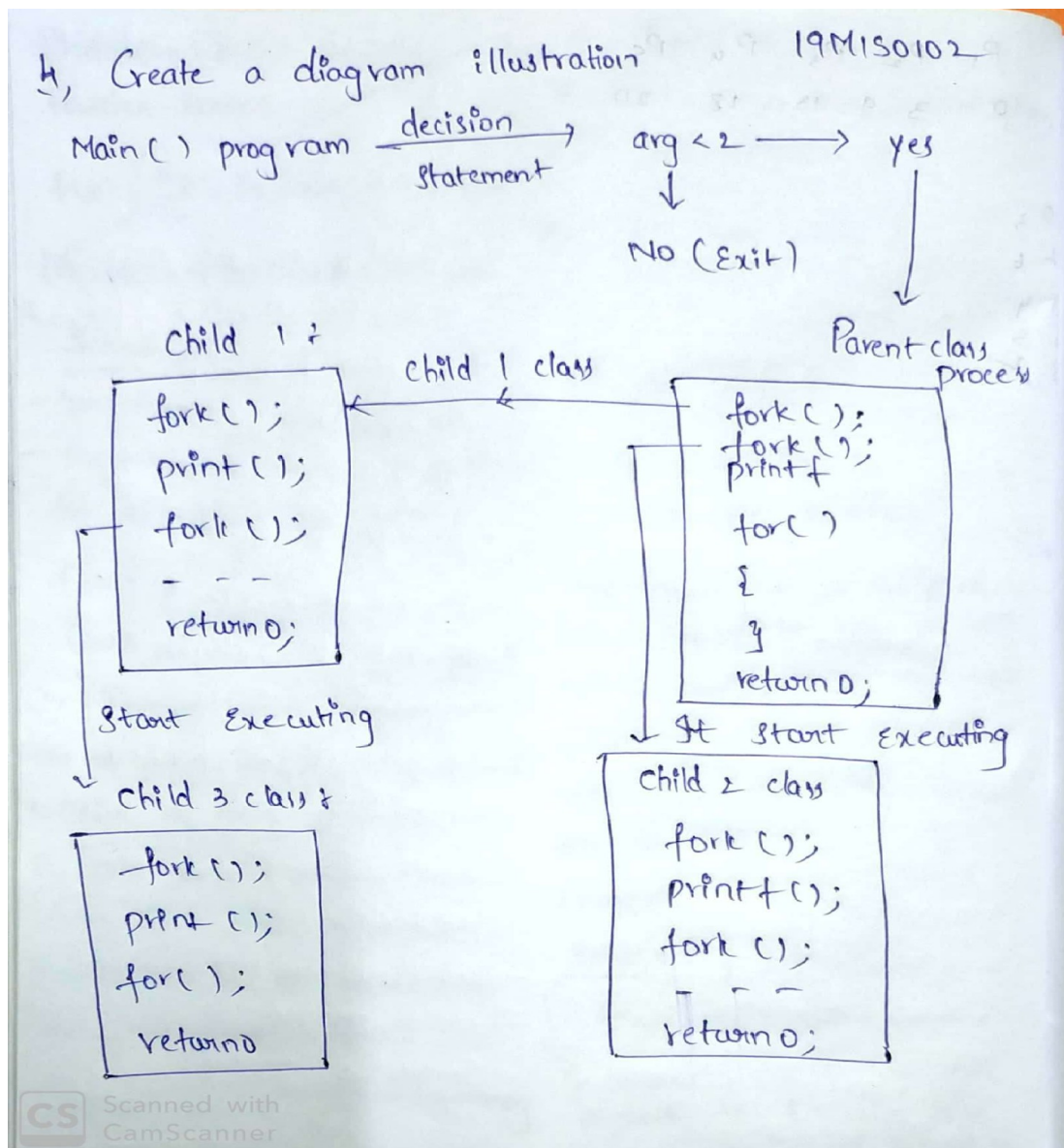
```

```

deepan2001@ubuntu:~/Desktop/Lab$ gcc Sample_Prog2.c
deepan2001@ubuntu:~/Desktop/Lab$ ./a.out 4
PID#: 2939
0
1
2
3
PID#: 2940
0
1
2
3
PID#: 2941
0
1
2
3
deepan2001@ubuntu:~/Desktop/Lab$ PID#: 2942
0
1
2
3
deepan2001@ubuntu:~/Desktop/Lab$

```

4. Create a diagram illustrating how Sample Program 2 executes (i.e. give a process hierarchy diagram, similar to the in-class exercise)



5. In the context of our classroom discussions on process state, process operations, and especially process scheduling, describe what you observed and try explain what is happening to produce the observed results.

```
deepan2001@ubuntu:~/Desktop/Lab$ gcc Sample_Prog2.c
deepan2001@ubuntu:~/Desktop/Lab$ ./a.out ps
PID#: 3729
PID#: 3730
deepan2001@ubuntu:~/Desktop/Lab$ PID#: 3731
PID#: 3732
```

- In this program, the first fork command creates a child process i.e. “Child 1” which contains the some lines of code to execute.
- Now The process Child 1 with the same code get executed and prints the output (Output: PID#3729).
- Now the control goes to main process which is the parent process. Here the second fork command is executed which will create another process “Child 2”.
- In the Child 2 process the lines are executed and prints the output PID#. (Output: PID#3720).
- Finally the control comes back to parent process and the further lines are executed and prints the output PID# . (Output: PID#3732)

Process Suspension and Termination :

Sample Program 3 :

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
```

```

int main()
{
    //use these variables
    pid_t pid,child;
    int status;
    if ((pid = fork()) < 0)
    {
        perror("fork failure");
        exit(1);
    }
    else if (pid ==0)
    {
        printf(" I am child PID %Id\n", (long) getpid());
    }
    else
    {
        printf("Child PID %Id terminated with return status %d\n", (long) child,
status);}

    return 0;}

```

```

deepan2001@ubuntu:~/Desktop/Lab$ gcc Sample_Prog3.c
Sample_Prog3.c: In function 'main':
Sample_Prog3.c:18:31: warning: format '%d' expects argument of type 'int', but argument 2 has type 'long int' [-Wformat=]
18 |     printf(" I am child PID %Id\n", (long) getpid());
    |                               ^~~~~~
    |                               |
    |                               int   long int
    |                               %Id
Sample_Prog3.c:22:29: warning: format '%d' expects argument of type 'int', but argument 2 has type 'long int' [-Wformat=]
22 |     printf("Child PID %Id terminated with return status %d\n", (long) child, status);
    |                               ^~~~~~
    |                               |
    |                               int   long int
    |                               %Id
deepan2001@ubuntu:~/Desktop/Lab$ ./a.out
Child PID 0 terminated with return status 0
I am child PID 3794
deepan2001@ubuntu:~/Desktop/Lab$

```

add function calls to Sample Program 3 so that it correctly uses wait() and exit(). Basically, implement the comments, making use of the pre-declared variables referenced in the printf() statement.

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
int main()
{
    //use these variables
    pid_t pid,child;
    int status;
    if ((pid = fork()) < 0)
    {
        perror("fork failure");
        exit(1);
    } else if (pid ==0) {
        printf(" I am child PID %Id\n", (long) getpid());
    }
    else {
        wait(&status);    /*applying signal function to the program */
        printf("Child PID %Id terminated with return status %d\n", (long) child,
status);
    }
    return 0; }
```

```
deepan2001@ubuntu:~/Desktop/Lab$ ./a.out
I am child PID 3851
Child PID 32766 terminated with return status 0
deepan2001@ubuntu:~/Desktop/Lab$
```

6. Provide the exact line of code that you inserted for the wait() system call.

```
wait(&status); /*applying signal function to the program */
```

7. Who prints first, the child or the parent? Why?

Child process prints first, the wait call returns the process id of the child process, which gives the parent the ability to wait for a particular child process to finish. The wait() call is done because the parent has to wait for the child process to complete its task.

8. What two values are printed out by the parent in Sample Program 3?

(No, not the actual numbers, but what they mean.) In other words, describe the interaction between the exit() function and the wait() system call. You may want to experiment by changing the value to better understand the interaction.

- The wait() causes the parent to wait for any child process. Exit() system call.
- The purpose of wait call() is to make the parent process process for the child processes exit() system call. When the child process want to exit () , it will send its status to parent to perform exit() action. For this, the

child process has to wait() until the child complete it's task . So, the wait() call is performed in the parent process.

- More generally, an exit in a multi- threading environment means that a thread of execution has stopped running

Process Execution:

Sample Program 4 :

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdlib.h>
int main(int argc, char* argv[])
{
if (argc < 2)
{
fputs("Usage : must supply a command\n",stderr);
exit(1);
}
puts("Before the exec");
if(execvp(argv[1], &argv[1]) < 0)
{
perror("exec failed");
exit(1);
}
puts("After the exec");
return 0;
}
```

```

deepan2001@ubuntu:~/Desktop/Lab$ gcc Sample_Prog4.c
deepan2001@ubuntu:~/Desktop/Lab$ ./a.out
Usage : must supply a command
deepan2001@ubuntu:~/Desktop/Lab$ ./a.out date
Before the exec
Mon 28 Feb 2022 09:09:09 AM PST
deepan2001@ubuntu:~/Desktop/Lab$ ./a.out ls
Before the exec
a.out demo.c New New.c Prog1.c Prog2.c Prog3.c Prog4.c Prog5.c Prog6.c Prog7.c Sampl
e_Prog1.c Sample_Prog2.c Sample_Prog3.c Sample_Prog4.c Test Test.c
deepan2001@ubuntu:~/Desktop/Lab$

```

Compile, run and test Sample Program 4 using various commands (e.g. "date", "ls")

9. When is the second print line ("After the exec") printed? Explain your answer.

The “After the exec” line is not printed. The line is not printed it may be due to error or the function call `exit()` is present before the execution of this line (“After the exec”). Because of the `exit()` function call, the second print line is not printed.

10. Explain how the second argument passed to `execvp()` is used?

`execvp()` returns a negative value if the execution fails. The second argument is a pointer to an array of character strings

`int main(int argc, char **argv) :`

When **`execvp()`** is executed, the program file given by the first argument will be loaded into the caller's address space and over-write the program there.

Then, the second argument will be provided to the program and starts the execution. As a result, once the specified program file starts its execution, the original program in the caller's address space is gone and is replaced by the new program.