**Name-Ujjawal Mandhani**

**Batch-F6**

**Enroll No-9918103237**

**OSSP – Lab 2**

**Question 1:** Write Linux C Program to display the process attributed as follows:

a. The process ID or PID

b. The parent process ID or PPID

c. Nice number

d. Terminal or TTY

e. User name of the real and effective user (RUID and EUID).

Sort the output of the above into files called procatt0.txtand procatt1.txt depending on

process creation time and CPU utilization time.

**Solution 1.a**

#include <stdio.h>

#include <unistd.h>

int main()

{

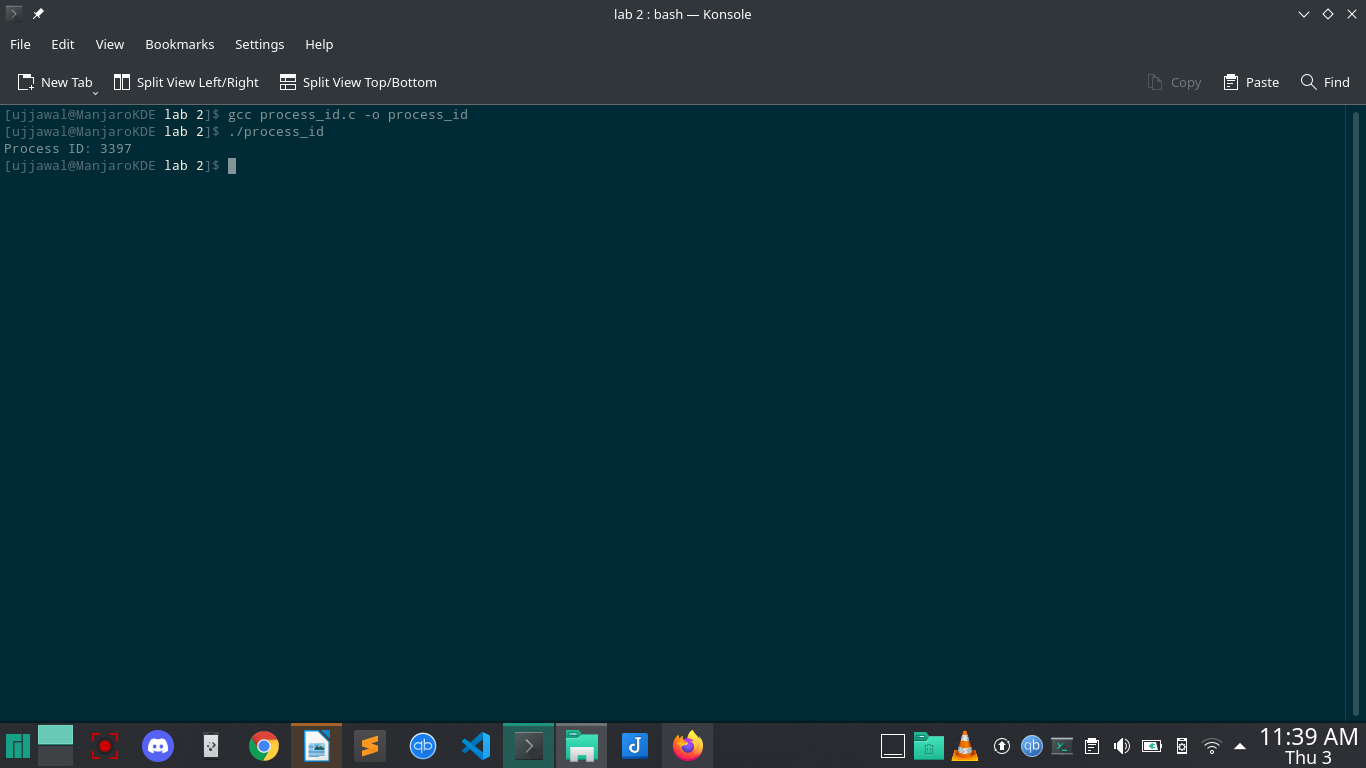
int p\_id;

p\_id=getpid(); /\*process id\*/

printf("Process ID: %d\n",p\_id);

return 0;

}



**Solution 1.b**

#include <stdio.h>

#include <unistd.h>

int main()

{

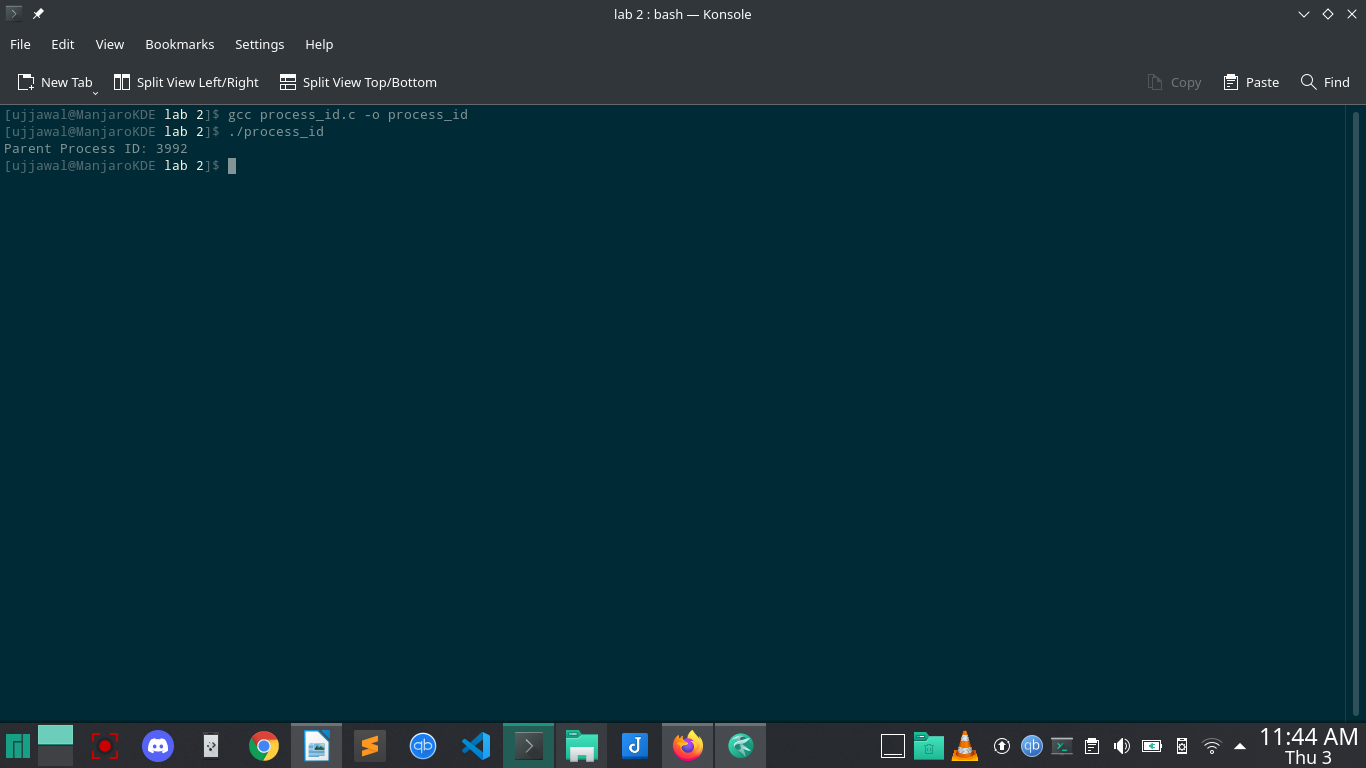
int p\_pid;

p\_pid=getpid(); /\*parent process id\*/

printf("Parent Process ID: %d\n",p\_pid);

return 0;

}



**Solution 1.c**

#include <stdio.h>

#include <unistd.h>

#include <sys/resource.h>

int main()

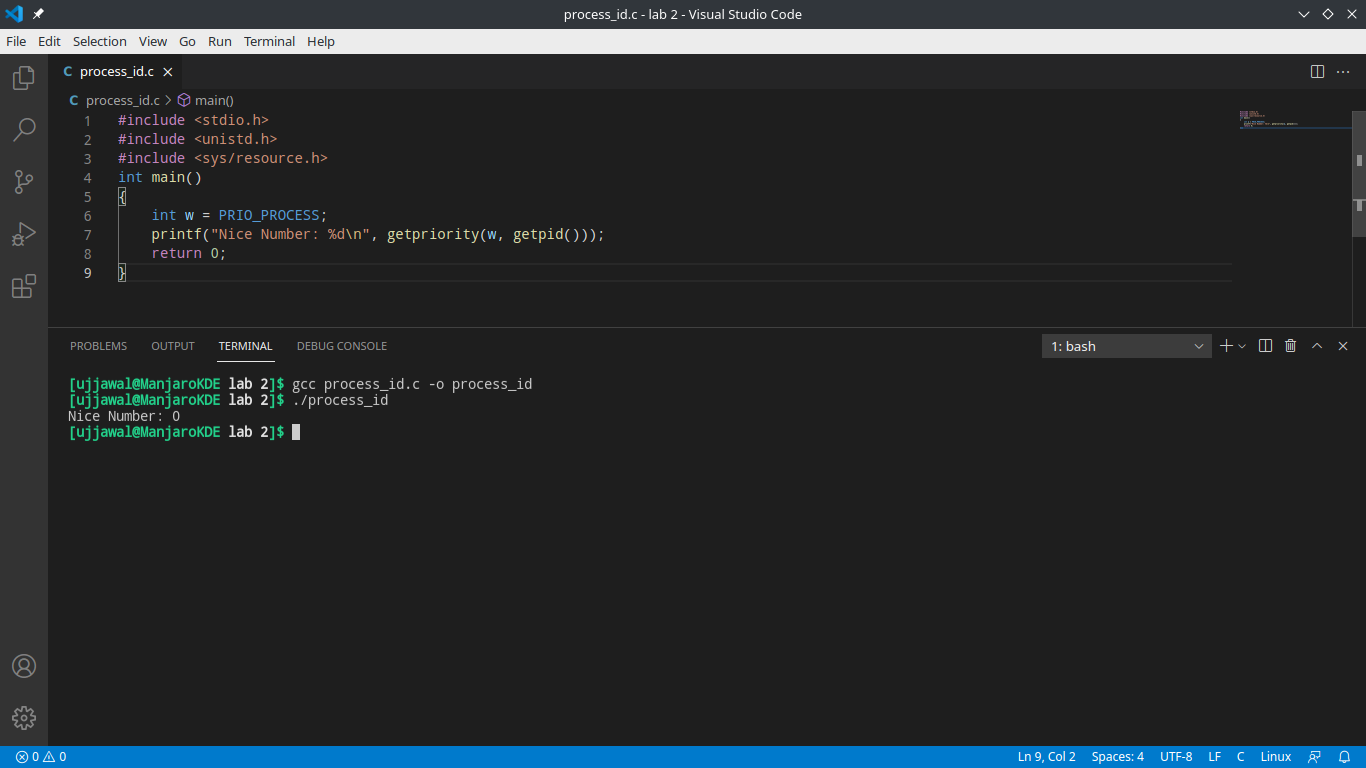
{

int w = PRIO\_PROCESS;

printf("Nice Number: %d\n", getpriority(w, getpid()));

return 0;

}



**Solution 1.d**

#include <unistd.h>

#include <stdio.h>

#include <stdlib.h>

int main(int argc, char \*\*argv)

{

int t, fd;

if (argc == 1)

{

printf("Syntax: %s 0,1,2\n", argv[0]);

exit(EXIT\_SUCCESS);

}

fd = atoi(argv[1]);

t = isatty(fd);

if (t)

printf("fd is %d, which is a tty\n", fd);

else

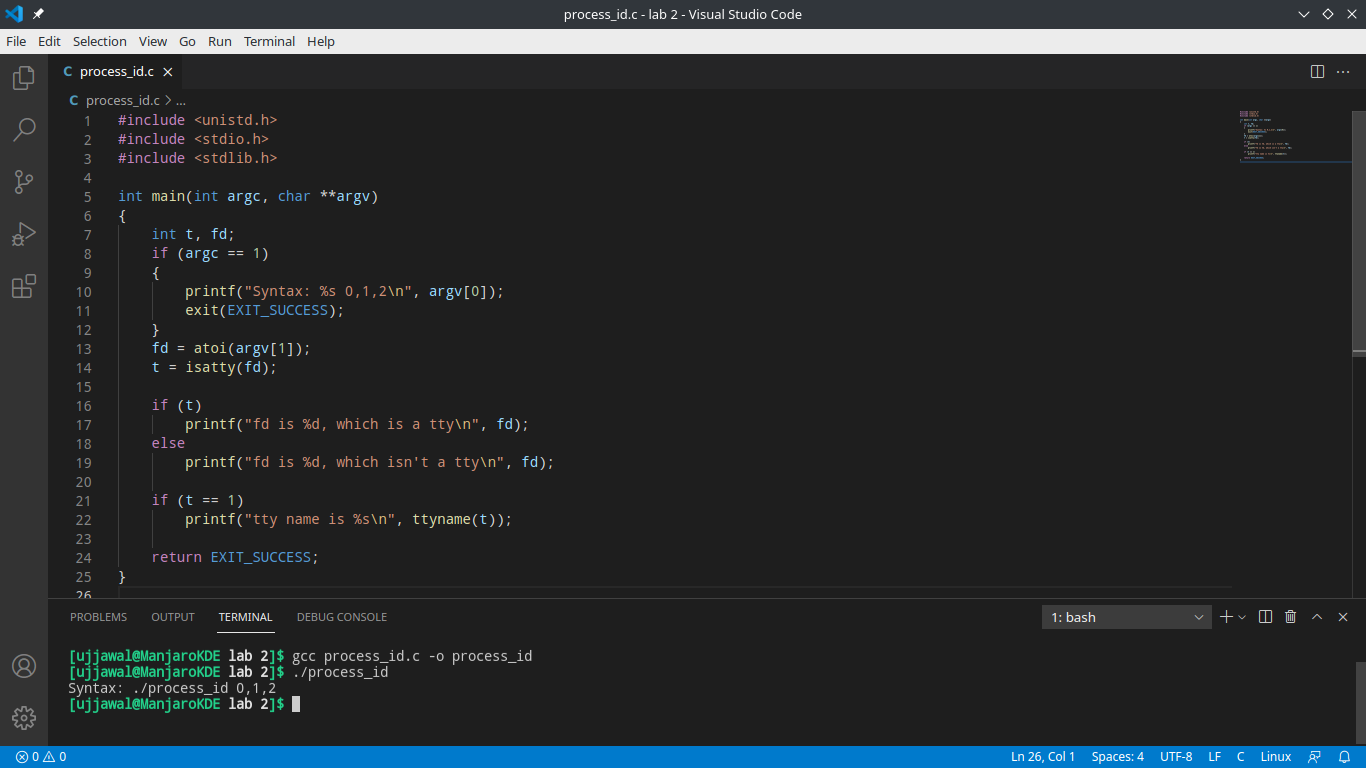
printf("fd is %d, which isn't a tty\n", fd);

if (t == 1)

printf("tty name is %s\n", ttyname(t));

return EXIT\_SUCCESS;

}



**Solution 1.e**

#include <stdio.h>

#include <unistd.h>

int main()

{

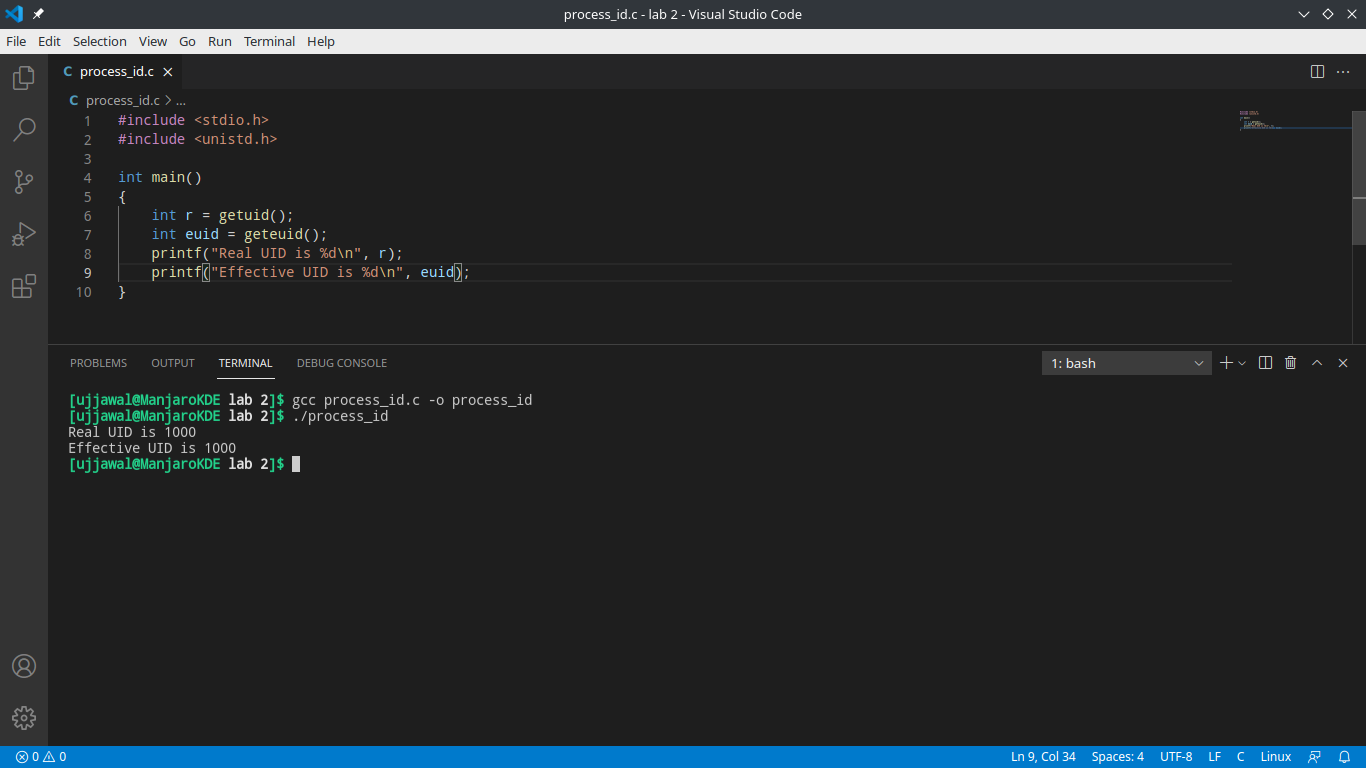
int r = getuid();

int euid = geteuid();

printf("Real UID is %d\n", r);

printf("Effective UID is %d\n", euid);

}



**Solution 2:**

#include <stdio.h>

#include <sys/types.h>

#include <unistd.h>

void forkexample()

{

int num = 10539;

printf("before fork: %d\n", num);

if (fork() == 0)

printf("I am the child %d\n", num + 1);

else

printf("I am the father %d\n", num, num + 1);

}

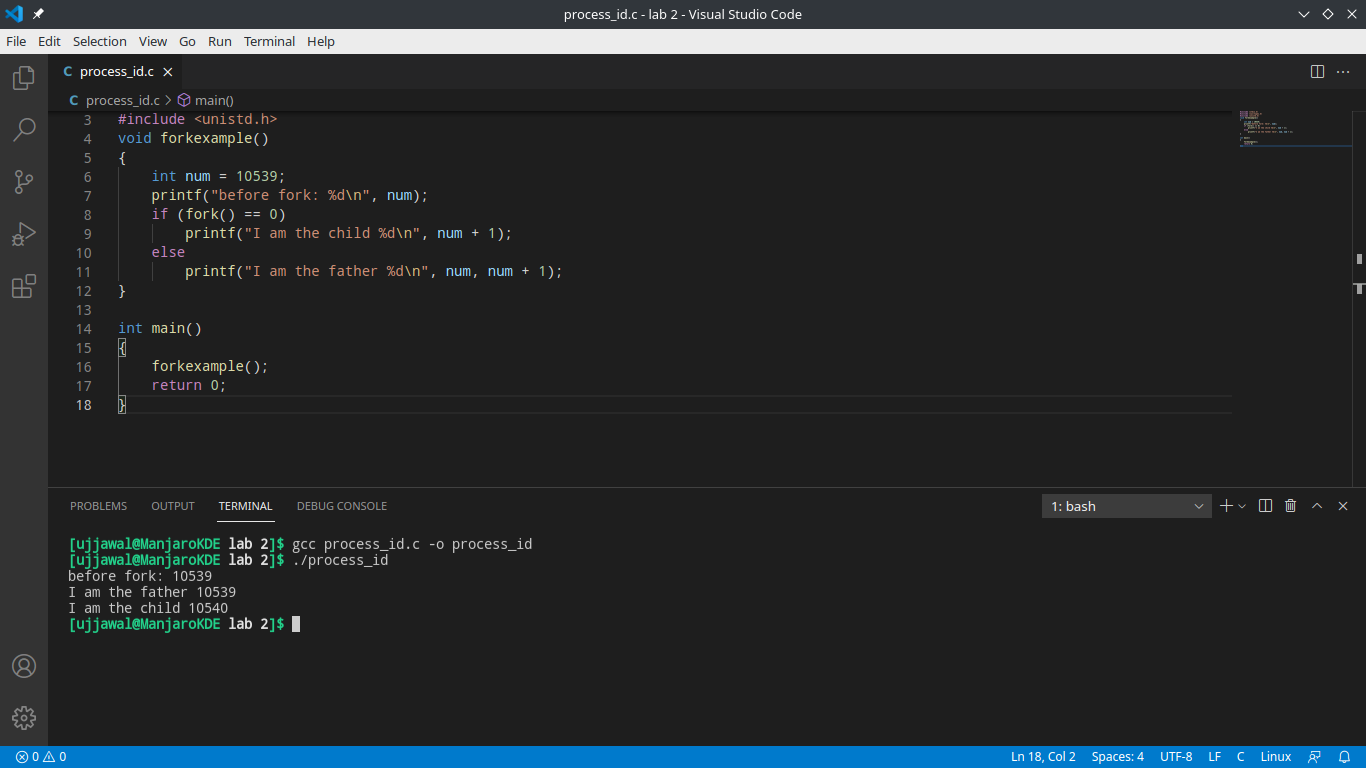
int main()

{

forkexample();

return 0;

}



**Solution 3:**

#include <stdio.h>

#include <stdlib.h>

#include <sys/types.h>

#include <unistd.h>

int main()

{

int pid\_t, child\_pid;

child\_pid = fork();

if (child\_pid > 0)

{

sleep(60);

}

else

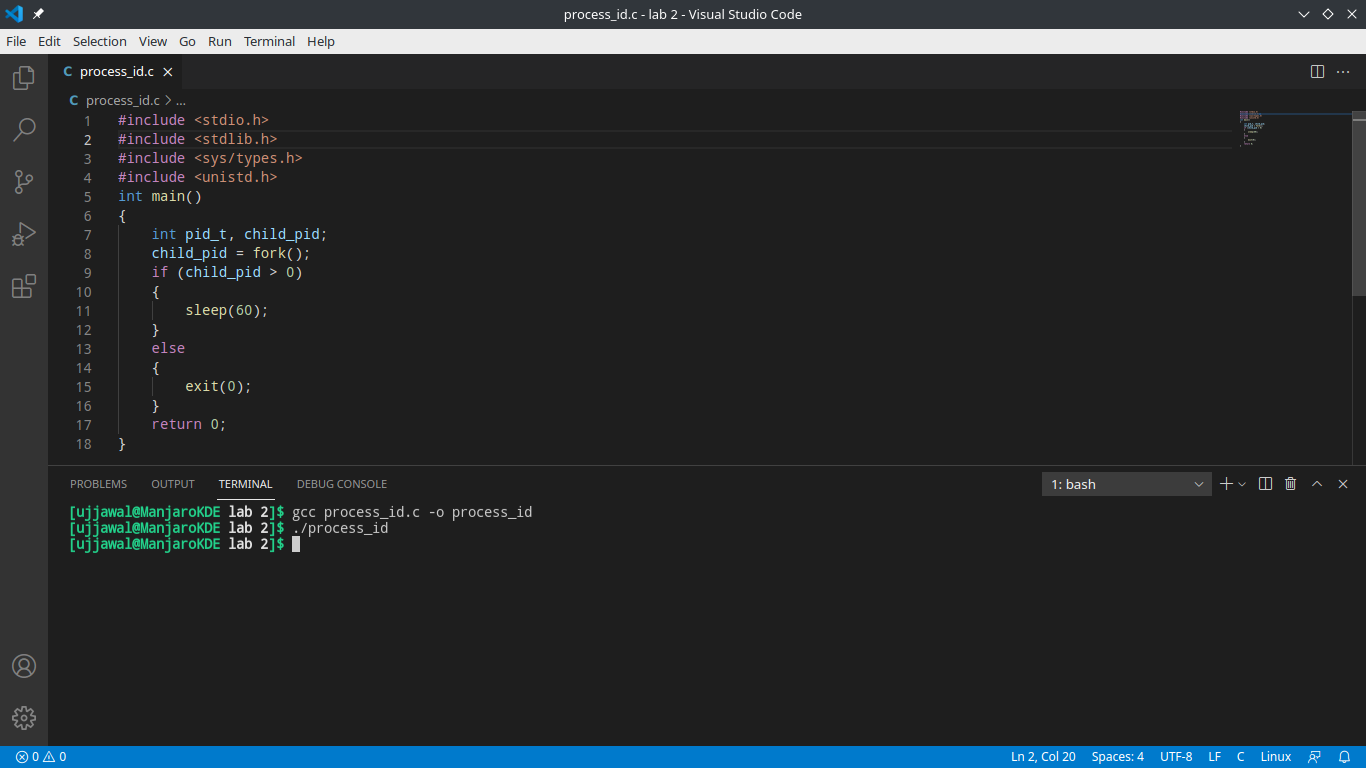
{

exit(0);

}

return 0;

}



**Solution 4:**

#include <stdio.h>

#include <stdlib.h>

#include <sys/types.h>

#include <unistd.h>

int main()

{

const pid\_t a = fork();

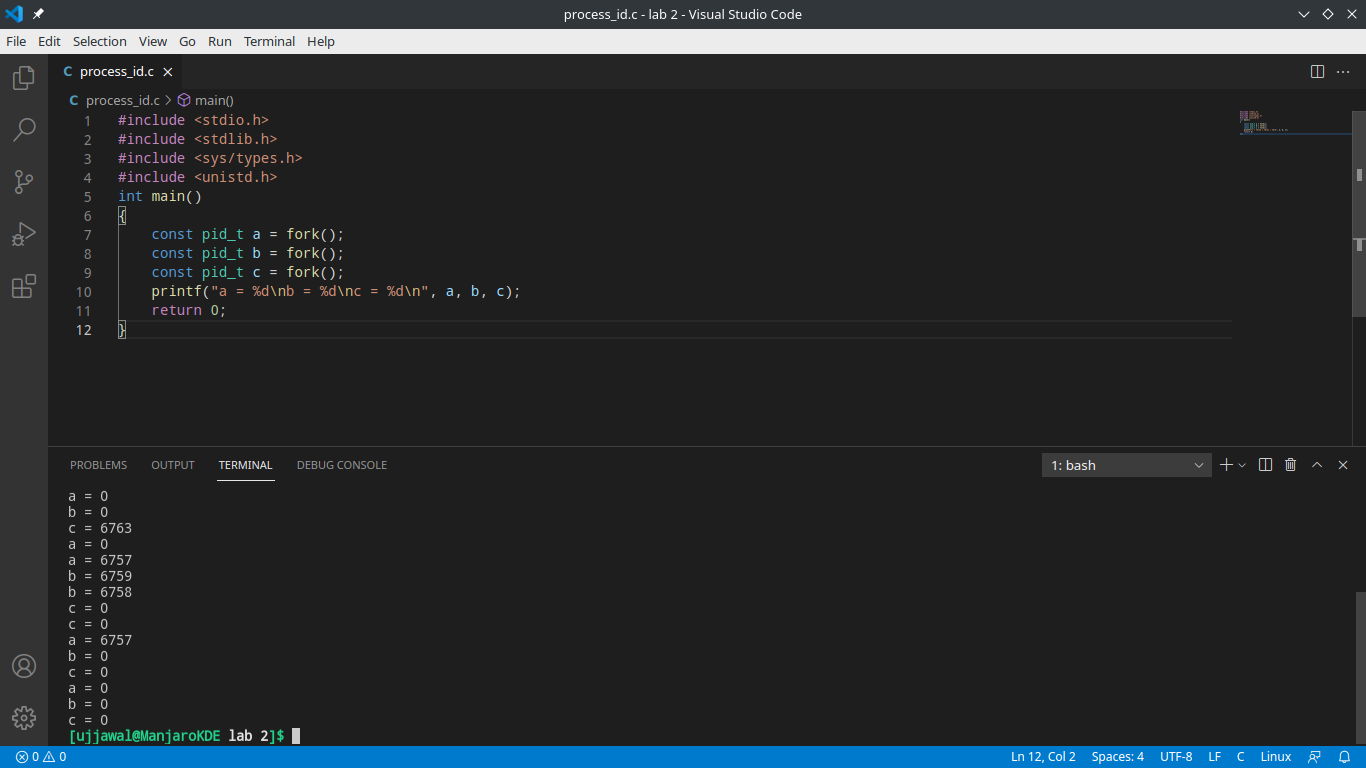
const pid\_t b = fork();

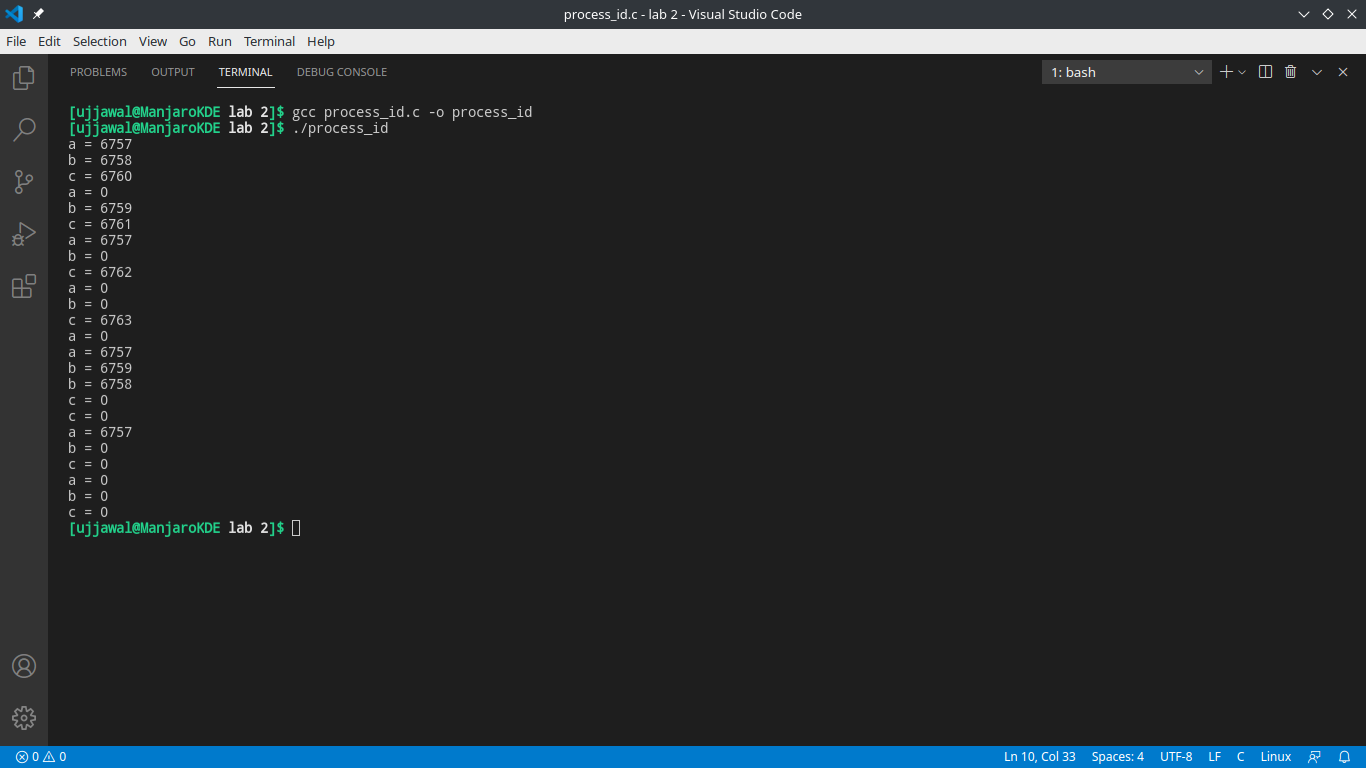
const pid\_t c = fork();

printf("a = %d\nb = %d\nc = %d\n", a, b, c);

return 0;

}

****



**Solution 5:**

#include <sys/types.h>

#include <sys/wait.h>

#include <errno.h>

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

int main(int argc, char \*argv[])

{

int i, pid, pid1, pid2, pid3, pid4, count;

int status;

if (argc != 3)

{

printf("incorrect number of arguments \n");

exit(EXIT\_FAILURE);

}

int A, B, S, rem;

count = 0;

S = 0;

A = atoi(argv[1]);

B = atoi(argv[2]);

printf("%d,\t %d\n", A, B);

pid1 = fork();

if (pid1 == 0)

{

pid = getpid();

printf("Forked child %d \n", pid);

S = A + B;

printf("I am child number %d with pid %d, the sum is %d \n", count + 1, pid, S);

exit(EXIT\_SUCCESS);

}

else if (pid1 < 0)

{

printf("Fork error %d.\n", errno);

exit(EXIT\_FAILURE);

}

pid2 = fork();

if (pid2 == 0)

{

pid = getpid();

printf("Forked child %d \n", pid);

S = A - B;

printf("I am child number %d with pid %d, the difference is %d \n", count + 1, pid, S);

exit(EXIT\_SUCCESS);

}

else if (pid2 < 0)

{

printf("Fork error %d.\n", errno);

exit(EXIT\_FAILURE);

}

pid3 = fork();

if (pid3 == 0)

{

pid = getpid();

printf("Forked child %d \n", pid);

S = A \* B;

printf("I am child number %d with pid %d, the product is %d \n", count + 1, pid, S);

exit(EXIT\_SUCCESS);

}

else if (pid3 < 0)

{

printf("Fork error %d.\n", errno);

exit(EXIT\_FAILURE);

}

pid4 = fork();

if (pid4 == 0)

{

pid = getpid();

printf("Forked child %d \n", pid);

S = A / B;

rem = A % B;

printf("I am child number %d with pid %d, the quotient is %d and the remainder is %d \n", count + 1, pid, S, rem);

exit(EXIT\_SUCCESS);

}

else if (pid4 < 0)

{

printf("Fork error %d.\n", errno);

exit(EXIT\_FAILURE);

}

for (i = 0; i < 3; ++i)

{

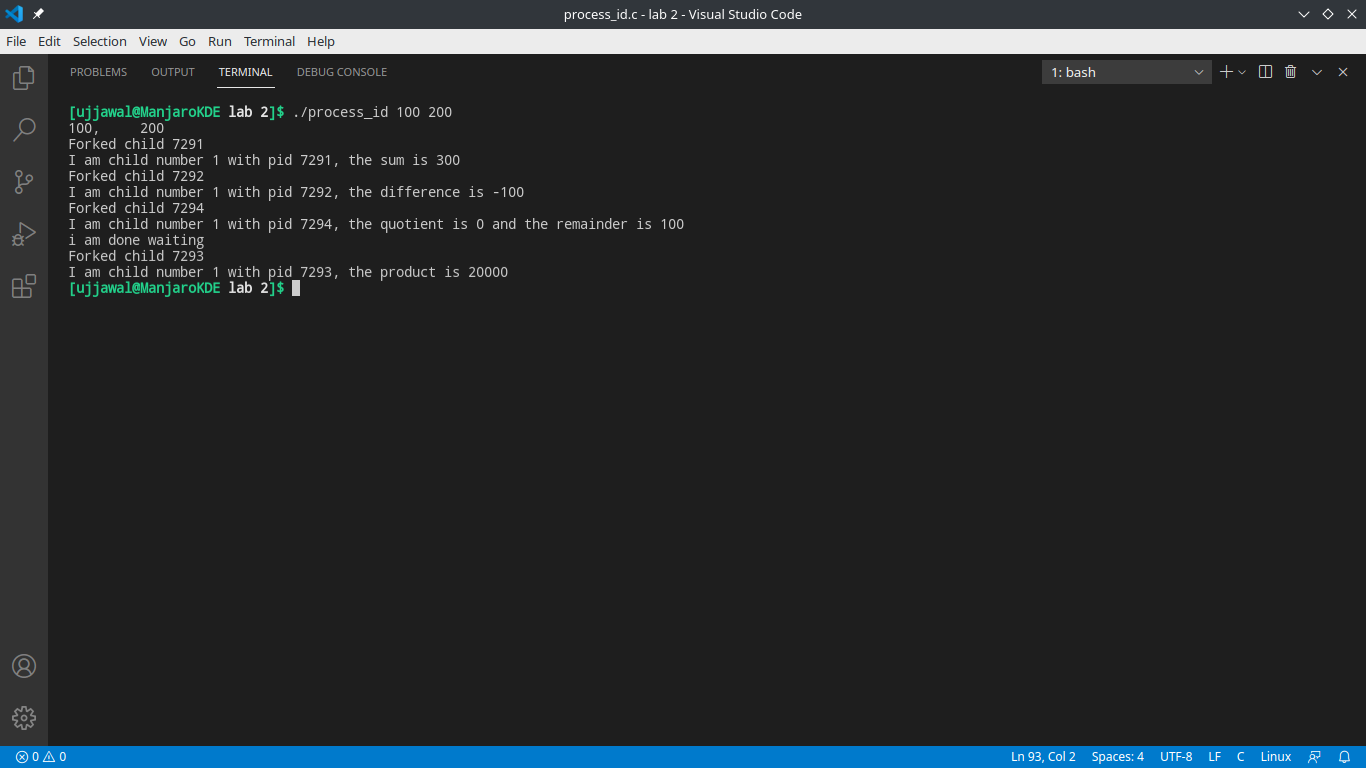
wait(&status);

}

printf("i am done waiting \n");

exit(EXIT\_SUCCESS);

}



**Solution 6:**

#include <sys/types.h>

#include <sys/wait.h>

#include <errno.h>

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

int main()

{

int pipefds[2];

int returnstatus;

int pid;

char writemessage[2][20] = {"Ujjawal", "Mandhani"};

char readmessage[20];

returnstatus = pipe(pipefds);

if (returnstatus == -1)

{

printf("Unable to create pipe\n");

return 1;

}

pid = fork();

if (pid == 0)

{

read(pipefds[0], readmessage, sizeof(readmessage));

printf("Child process - Reading from pipe - Message 1 is %s\n", readmessage);

read(pipefds[1], readmessage, sizeof(readmessage));

printf("Child process - Reading from pipe - Message 2 is %s\n", readmessage);

}

else

{

printf("Parent process - Reading from pipe - Message 1 is %s\n", writemessage[0]);

write(pipefds[0], writemessage[0], sizeof(readmessage));

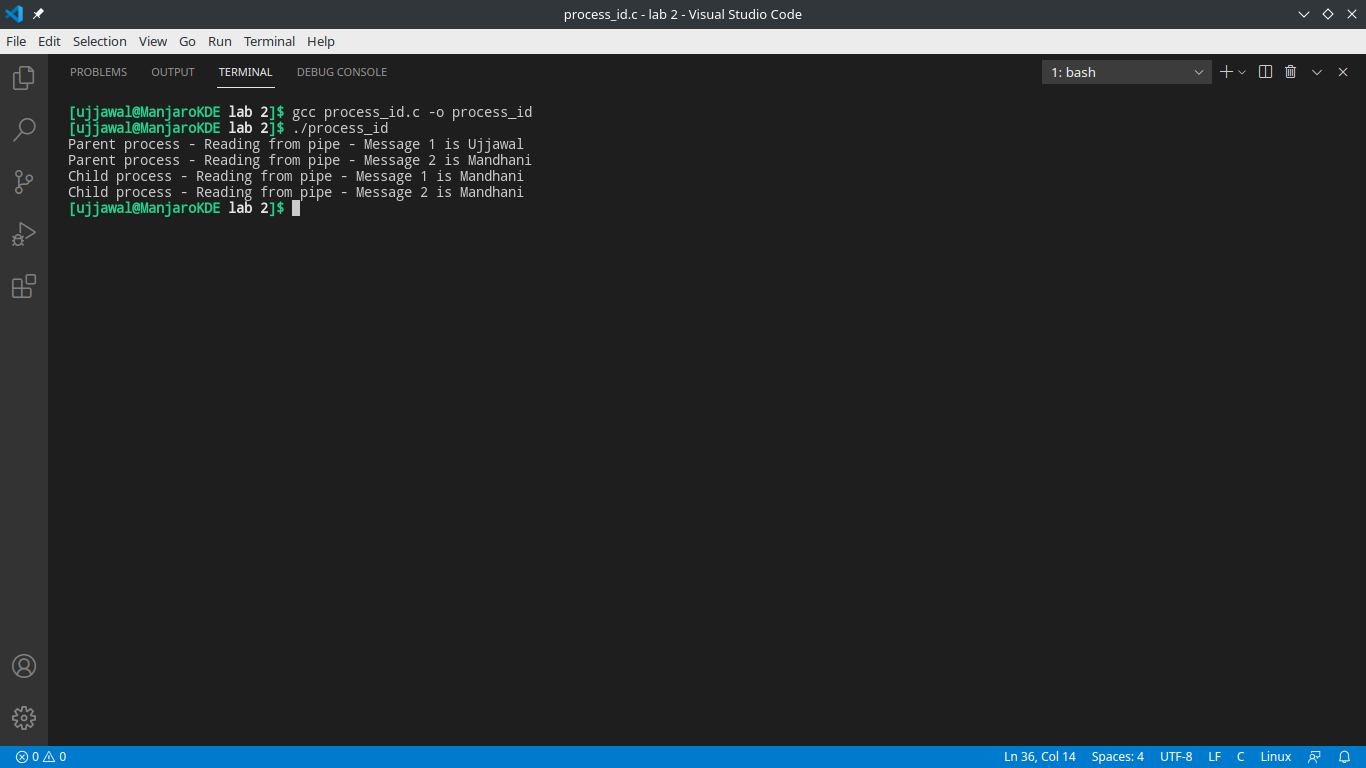
printf("Parent process - Reading from pipe - Message 2 is %s\n", writemessage[1]);

write(pipefds[1], writemessage[1], sizeof(readmessage));

}

return 0;

}

****