**4. WAP (using fork() and/or exec() commands) where parent and child execute:**

**a. same program, same code**

**b. same program, different code**

**c. different programs**

**d. before terminating, the parent waits for the child to finish its task**

(a) #include<sys/times.h>

#include<stdio.h>

#include<unistd.h>

#include<sys/wait.h>

int main()

{

pid\_t pid;

pid=fork();

if(pid<0)

{

fprintf(stderr, "Fork Failed");

return 1;

}

else

{

printf("Child ID: %d\n", pid);

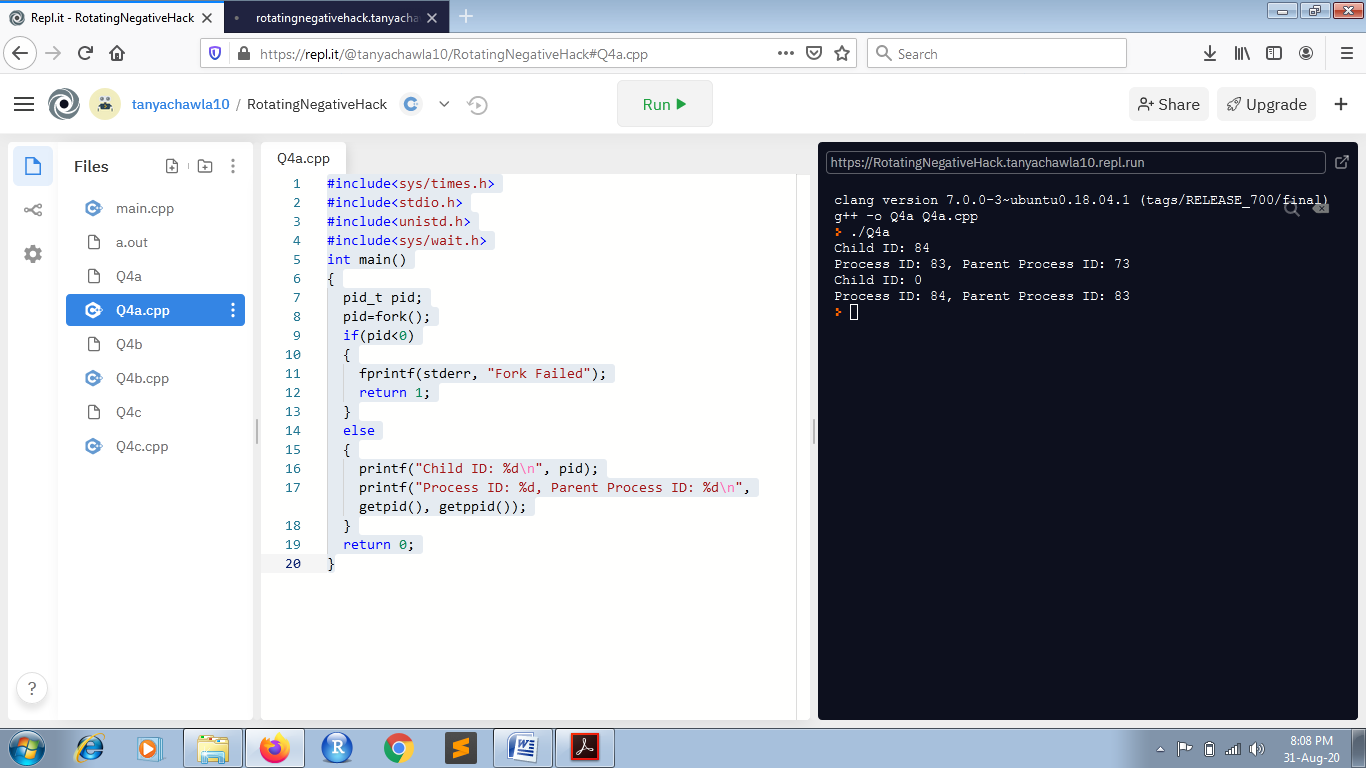
printf("Process ID: %d, Parent Process ID: %d\n", getpid(), getppid());

}

return 0;

}

**OUTPUT**

****

(b) #include<sys/types.h>

#include<stdio.h>

#include<unistd.h>

#include<sys/wait.h>

int main()

{

pid\_t pid;

pid=fork();

if(pid<0)

{

printf("Fork Failed");

return 1;

}

else if(pid==0)

{

printf("Child Process\n");

}

else

{

wait(NULL);

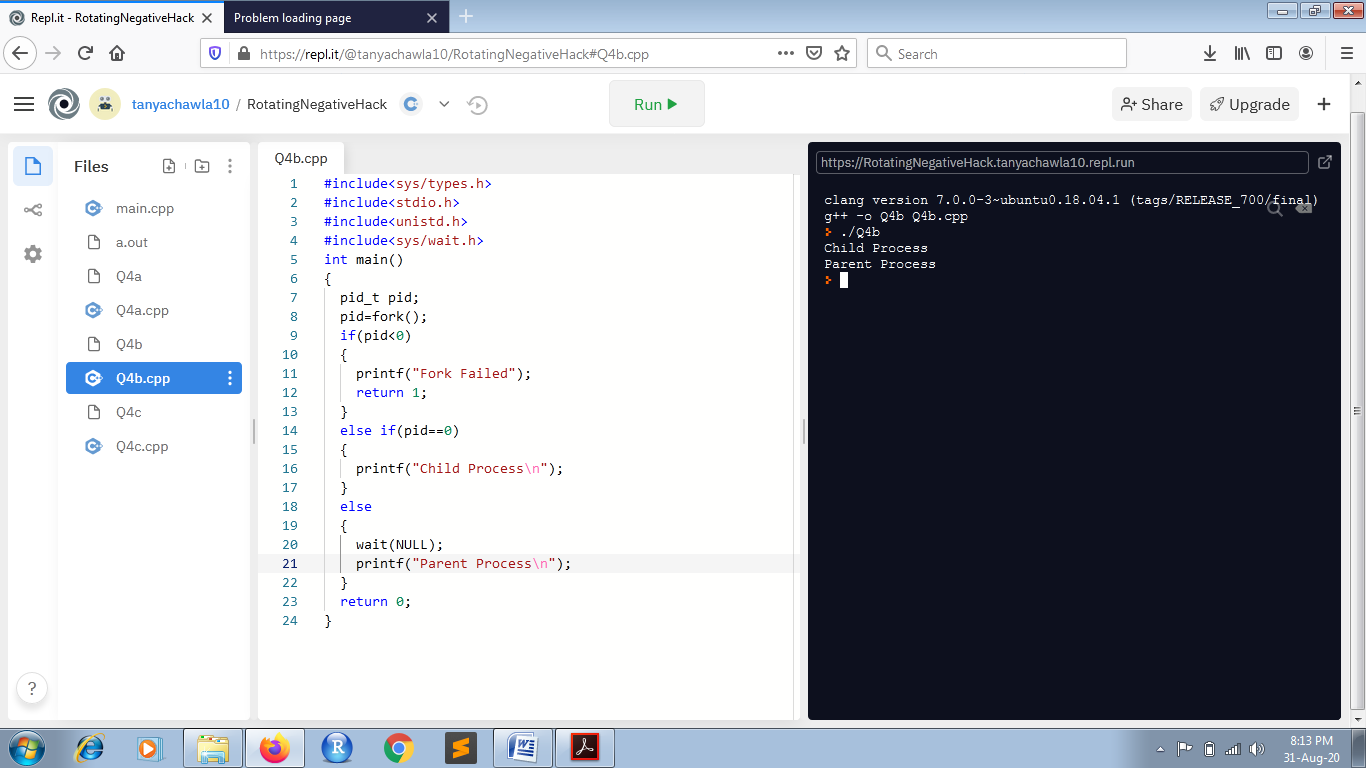
printf("Parent Process\n");

}

return 0;

}

**OUTPUT**

****

(c) #include<sys/times.h>

#include<stdio.h>

#include<unistd.h>

#include<sys/wait.h>

int main()

{

pid\_t pid;

pid=fork();

if(pid<0)

{

fprintf(stderr, "Fork Failed");

return 1;

}

if(pid==0)

{

execlp("/home/runner/RotatingNegativeHack/main", "main", NULL);

}

else

{

wait(NULL);

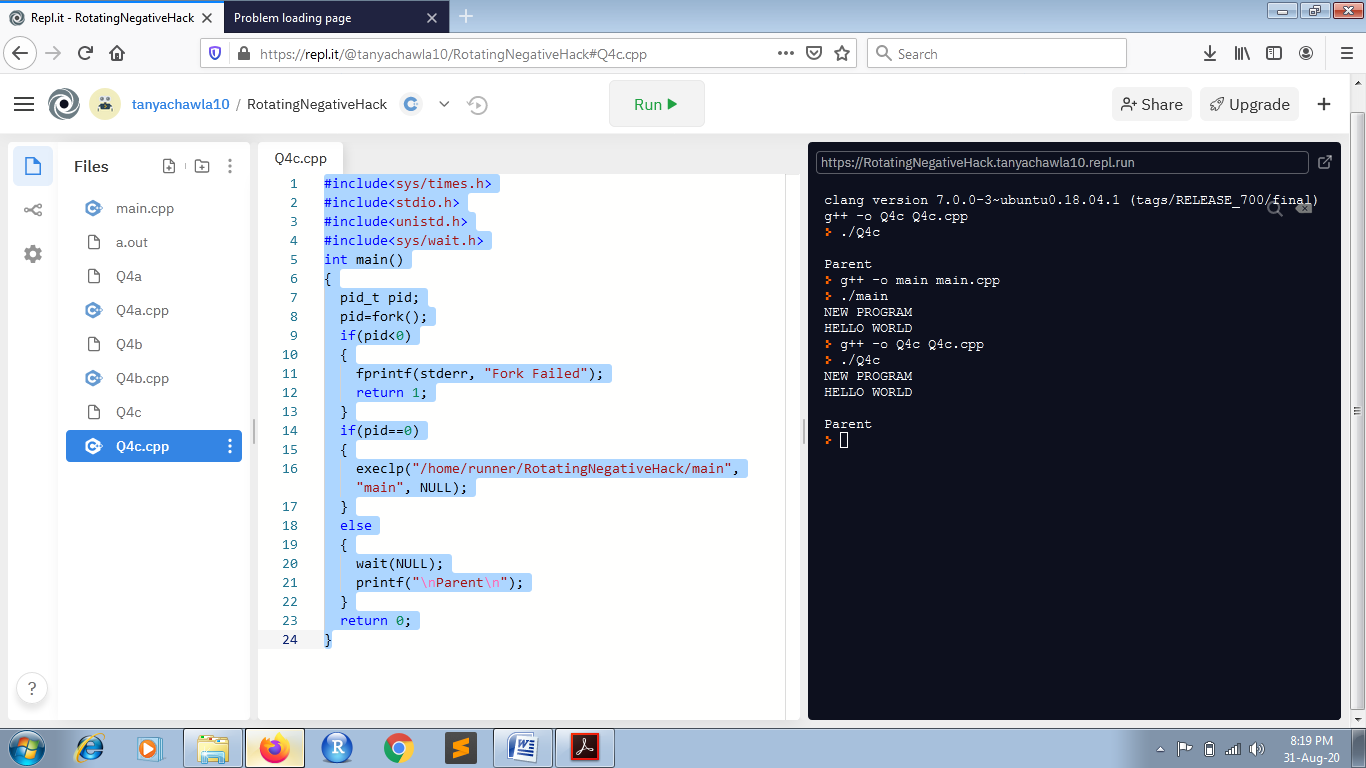
printf("\nParent\n");

}

return 0;

}

**OUTPUT**

****