**Autocorrezione Esercizio Fork()**

#include<sys/types.h>

#include<stdio.h>

#include<stdlib.h>

#include<fcntl.h>

#include<signal.h>

#include<wait.h>

#include<sys/wait.h>

#include<unistd.h>

int main()

{

int pid;

int in\_fd, out\_fd;

int status, n, contatore=0;

char buff;

pid = fork(); //Creazione del figlio

if(pid<0) printf("Errore nella fork \n");

if(pid==0)

{

signal(SIGINT, SIG\_IGN); //Ignora il segnale immesso da input

/\* Apertura del file \*/

in\_fd=open("PIPPO.txt", O\_RDONLY);

out\_fd=open("TOPOLINO.txt", O\_CREAT|O\_WRONLY, 0600);

while((n=read(in\_fd,&buff,1))>0)

{

if(buff!='A'&&buff!='a'&&buff!='E'&&buff!='e'&&buff!='I'&&buff!='i'&&buff!='O'&&buff!='o'&&buff!='U'&&buff!='u')

{

contatore=contatore+1;

if(write(out\_fd,&buff,1)!=1)

{

printf("Errore nella write \n");

}

}

}

printf("Il numero di consonanti scritte e': %d \n", contatore);

/\* Chiusura del file \*/

close(in\_fd);

close(out\_fd);

exit(1);

}

else if(pid>0)

{

signal(SIGINT, SIG\_IGN); //Ignora il segnale immesso da input

wait(&status);

printf("Il pid di mio figlio e': %d \n", getpid());

printf("Il processo padre termina \n");

exit(1);

}

return 0;

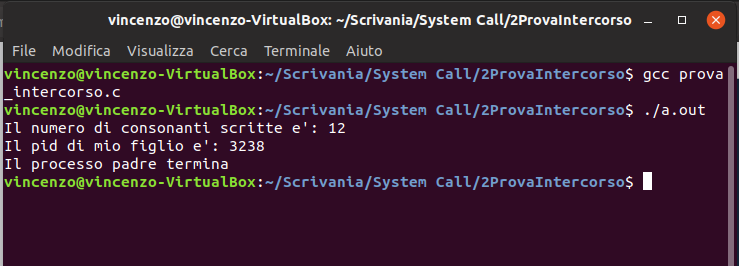
}

**Descrizione Modifiche**

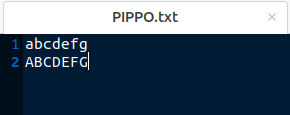
Per il corretto funzionamento del codice sono state apportate le seguenti modifiche:

* Ho aggiungo la libreria #include<sys/wait.h > e la libreria #include<unistd.h>.
* Ho aggiunto l’argomento oflag O\_WRONLY.
* Ho corretto qualche errore di sintassi commesso per distrazione.

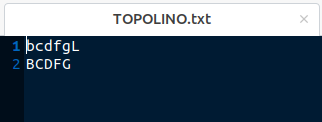
**Simulazione esecuzione del programma**



File input:



File output:



**Descrizione 2° programma**

Ho deciso di migliorare il programma precedente creando un'altra soluzione più efficiente e più corretta possibile. Il programma avrà le stesse funzionalità del programma precedente con l’aggiunta del controllo dei caratteri speciali.

Per ottimizzare i controlli fatti nel precedente programma, ho deciso di inserire un vettore di caratteri, il quale indicherà solo ed esclusivamente i caratteri che sono ammessi nel file di output. Quindi con questo programma è possibile distinguere i caratteri normali dai caratteri speciali con adeguati controlli.

#include<sys/types.h>

#include<sys/wait.h>

#include<stdio.h>

#include<stdlib.h>

#include<fcntl.h>

#include<signal.h>

#include<unistd.h>

#include<ctype.h>

int main()

{

int pid;

int in\_fd, out\_fd;

int status, n, contatore=0;

char buff;

char consonanti[]={'b','c','d','f','g','h','l','m','n','p','q','r','s','z','x','t','y','w'};

pid = fork(); //Creazione del figlio

if(pid<0) printf("Errore nella fork \n");

if(pid==0)

{

signal(SIGINT, SIG\_IGN); //Ignora il segnale immesso da input

in\_fd=open("PIPPO.txt", O\_RDONLY);

out\_fd=open("TOPOLINO.txt", O\_CREAT|O\_WRONLY|O\_TRUNC, 0600);

while((n=read(in\_fd,&buff,1))>0)

{

for(int i=0;consonanti[i]!='\0';i++)

{

/\* Con il controllo seguente sto distinguendo i caratteri normali dai caratteri speciali. Esso infatti permette di controllare se il carattere preso dal file faccia parte del vettore consonanti[i] \*/

if((buff==consonanti[i])||buff==toupper(consonanti[i]))

{

contatore++;

if(write(out\_fd,&buff,1)!=1)

{

printf("Errore nella write \n");

}

}

}

}

printf("Il numero di consonanti scritte e': %d \n", contatore);

close(in\_fd);

close(out\_fd);

exit(1);

}

else if(pid>0)

{

signal(SIGINT, SIG\_IGN);

wait(&status);

printf("Il pid di mio figlio e': %d \n", getpid());

printf("Il processo padre termina \n");

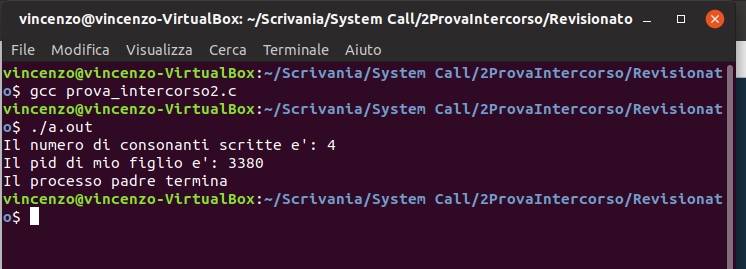
exit(1);

}

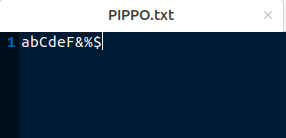
return 0;

}

**Simulazione esecuzione del programma**



File input:



File output:

