**INHA UNIVERSITY TASHKENT**

**DEPARTMENT OF CSE & ICE**

**SPRING SEMESTER 2017**

**SOC 2040 - SYSTEMS PROGRAMMING**

**LAB ASSIGNMENT 5**

**Submitted by**

**Student Name: Maftuna Sharabbaeva**

**Student ID U1510067**

**Group : 002 Sophomore ICE**



**INSTRUCTIONS :**

**- All Lab assignments are to be completed by students in a group of maximum three students**

**- Screen shots are to be provided wherever necessary**

**- Lab Assignment Report should be prepared using the Template provided and uploaded at the E-class portal by each student separately**

**- In the report, for each question you need to provide Algorithm, Program and Results Screenshots.**

**- Source code for all the programs should be uploaded along with the Lab Report at the E-Class portal.**

**- Last date for submission of the Lab Assignment is 1st May 2017**

**- Late submissions not entertained, Adhere to the deadline strictly**

**- All programs required for questions 3, 4 and 5 of this lab assignment are given in a folder sp5programs**

**Q1. Answer:**

I question 1 firstly we should write code for matrix addition, matrix subtraction and matrix multiplication and matrix transposition. Here they are :

**Code for matrix addition:**

#include<stdio.h>

void matadd(int \*\*x, int \*\*y, int \*\*z, int n)

{

for(int i = 0; i < n; i++)

{

for(int j = 0; j < n; j++)

{

z[i][j] = x[i][j] + y[i][j];

}

}

}

**Code for matrix subtraction:**

#include<stdio.h>

void matsub(int \*\*x, int \*\*y, int \*\*z, int n)

{

for(int i = 0; i < n; i++)

{

for(int j = 0; j < n; j++)

{

z[i][j] = x[i][j] - y[i][j];

}

}

}

**Code for matrix multiplication:**

#include<stdio.h>

void matmult(int \*\*x, int \*\*y, int \*\*z, int n)

{

for(int i = 0; i < n; i++)

{

for(int j = 0; j < n; j++)

{

z[i][j] = x[i][j] \* y[i][j];

}

}

}

**Code for matrix transposing:**

#include<stdio.h>

void mattrans(int \*\*y, int \*\*z, int n)

{

for(int i = 0; i < n; i++)

{

for(int j = 0; j < n; j++)

{

z[i][j] = y[j][i];

}

}

}

I wrote two main functions separately. MainFunc is for running at comile tima , and at Run time. But MainFunc1 is for running at load time. Here their codes:

**Code of MainFunc:**

#include<string.h>

#include<math.h>

#include <stdio.h>

#include <stdlib.h>

#include<dlfcn.h>

/\* we reverse space through malloc function \*/

int \*\*readMatrix( char\* fileName, int\* n)

{

FILE\* file\_ptr = fopen(fileName, "r");

fscanf(file\_ptr, "%d", n);

int \*\*result = (int\*\*)malloc(\*n\*sizeof(int\*));

for(int i = 0; i < \*n; i++)

{

result[i] = (int\*)malloc(\*n\*sizeof(int));

for(int j = 0; j <\*n; j++)

{

fscanf(file\_ptr, "%d", &result[i][j]);

}

}

fclose(file\_ptr);

return result;

}

/\* yousee here free space what we did by malloc function \*/

void myFree(int \*\*matrix, int n)

{

for(int i = 0; i < n; i++)

{

free(matrix[i]);

}

free(matrix);

}

/\* i build matrix via malloc function \*/

int \*\*createMatrix(int n)

{

int \*\*result = (int\*\*)malloc(n\*sizeof(int\*));

for(int i = 0; i < n; i++)

{

result[i] = (int\*)malloc(n\*sizeof(int));

for(int j = 0; j < n; j++)

{

result[i][j] = 0;

}

}

return result;

}

/\* we write new matrix into file \*/

void writeMatrix(char \*fileName, int\*\*z, int n)

{

FILE \*file\_ptr = fopen(fileName, "w");

for(int i=0; i<n; i++)

{

for(int j=0; j<n; j++)

fprintf(file\_ptr, "%d ", z[i][j]);

fprintf(file\_ptr, "\n");

}

fclose(file\_ptr);

}

void matadd(int \*\*x, int \*\*y, int \*\*z, int n);

void matsub(int \*\*x, int \*\*y, int \*\*z, int n);

void matmult(int \*\*x, int \*\*y, int \*\*z, int n);

void mattrans(int \*\*y, int \*\*z, int n);

int main()

{

int n = 0;

int\*\* x = readMatrix("MATRIX2X2.txt", &n);

int\*\* y = readMatrix("MATRIX2X2B.txt", &n);

int\*\* z = createMatrix(n);

matadd(x,y,z,n);

writeMatrix("addMATRIX2X2.txt", z, n);

myFree(z, n);

z = createMatrix(n);

matsub(x,y,z,n);

writeMatrix("subMATRIX2X2.txt", z, n);

myFree(z, n);

z = createMatrix(n);

matmult(x,y,z,n);

writeMatrix("multMATRIX2X2.txt", z, n);

myFree(z, n);

z = createMatrix(n);

mattrans(y,z,n);

writeMatrix("transMATRIX2X2.txt", z, n);

myFree(z, n);

myFree(x, n);

myFree(y, n);

printf("Well done! Complete :) \n");

}

**Code for MainFunc1:**

#include<string.h>

#include<math.h>

#include <stdio.h>

#include <stdlib.h>

#include<dlfcn.h>

/\*we reverse space via malloc function \*/

int \*\*readMatrix( char\* fileName, int\* n)

{

FILE\* file\_ptr = fopen(fileName, "r");

fscanf(file\_ptr, "%d", n);

int \*\*result = (int\*\*)malloc(\*n\*sizeof(int\*));

for(int i = 0; i < \*n; i++)

{

result[i] = (int\*)malloc(\*n\*sizeof(int));

for(int j = 0; j <\*n; j++)

{

fscanf(file\_ptr, "%d", &result[i][j]);

}

}

fclose(file\_ptr);

return result;

}

/\* malloc built free spaces \*/

void myFree(int \*\*matrix, int n)

{

for(int i = 0; i < n; i++)

{

free(matrix[i]);

}

free(matrix);

}

/\* Building matrix through malloc \*/

int \*\*createMatrix(int n)

{

int \*\*result = (int\*\*)malloc(n\*sizeof(int\*));

for(int i = 0; i < n; i++)

{

result[i] = (int\*)malloc(n\*sizeof(int));

for(int j = 0; j < n; j++)

{

result[i][j] = 0;

}

}

return result;

}

/\*We write new matrix into file here :) \*/

void writeMatrix(char \*fileName, int\*\*z, int n)

{

FILE \*file\_ptr = fopen(fileName, "w");

for(int i=0; i<n; i++)

{

for(int j=0; j<n; j++)

fprintf(file\_ptr, "%d ", z[i][j]);

fprintf(file\_ptr, "\n");

}

fclose(file\_ptr);

}

void matadd(int \*\*x, int \*\*y, int \*\*z, int n);

void matsub(int \*\*x, int \*\*y, int \*\*z, int n);

void matmult(int \*\*x, int \*\*y, int \*\*z, int n);

void mattrans(int \*\*y, int \*\*z, int n);

int main()

{

int n = 0;

void \*handle;

void (\*matadd)(int\*\*, int\*\*, int\*\*, int);

void (\*matsub)(int\*\*, int\*\*, int\*\*, int);

void (\*matmult)(int\*\*, int\*\*, int\*\*, int);

void (\*mattrans)(int\*\*, int\*\*, int);

char\* error;

handle = dlopen("./libmatrix.so", RTLD\_LAZY);

if(!handle)

{

fprintf(stderr, "%s\n", dlerror());

exit(1);

}

matadd = dlsym(handle, "matadd");

if((error = dlerror())!=NULL)

{

fprintf(stderr, "%s\n", error);

exit(1);

}

matsub = dlsym(handle, "matsub");

if((error = dlerror())!=NULL)

{

fprintf(stderr, "%s\n", error);

exit(1);

}

matmult = dlsym(handle, "matmult");

if((error = dlerror())!=NULL)

{

fprintf(stderr, "%s\n", error);

exit(1);

}

mattrans = dlsym(handle, "mattrans");

if((error = dlerror())!=NULL)

{

fprintf(stderr, "%s\n", error);

exit(1);

}

int\*\* x = readMatrix("MATRIX2X2.txt", &n);

int\*\* y = readMatrix("MATRIX2X2B.txt", &n);

int\*\* z = createMatrix(n);

matadd(x,y,z,n);

writeMatrix("addMat2x2.txt", z, n);

myFree(z, n);

z = createMatrix(n);

matsub(x,y,z,n);

writeMatrix("subMat2x2.txt", z, n);

myFree(z, n);

z = createMatrix(n);

matmult(x,y,z,n);

writeMatrix("multMat2x2.txt", z, n);

myFree(z, n);

z = createMatrix(n);

mattrans(y,z,n);

writeMatrix("transposeMat2x2.txt", z, n);

myFree(z, n);

myFree(x, n);

myFree(y, n);

printf("Action completed!!!\n");

if(dlclose(handle)<0){

fprintf(stderr, "%s\n", dlerror());

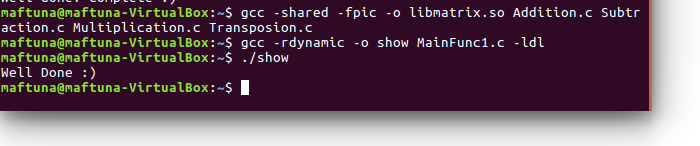
exit(1);

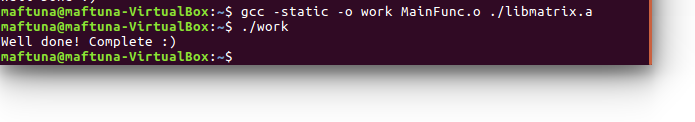
}

return 0;

}

I created Library Archive libmatrix.a for static linking and sharable library libmatrix.so for dynamic linking. In screenshts you can see their codes. In addition I have some text files you can see them in separated files with C codes of above functions! 





**Answer for question 2:**

1. Codes for malloc and free functions to carry out inter-positioning at compile time:

**int.c :**

#include <stdio.h>

#include <malloc.h>

int main()

{

int \*p=malloc(32);

free(p);

check();

return(0);

}

**malloc.c:**

#define malloc(size) mymalloc(size)

#define free(ptr) myfree(ptr)

void \*mymalloc(size\_t size);

void myfree(void \*ptr);

**mymalloc.c:**

#ifdef COMPILETIME

#include <stdio.h>

#include <malloc.h>

int n=0, m=0;

void \*mymalloc(size\_t size)

{

void \*ptr=malloc(size);

printf("malloc(%d)=%p\n", (int)size, ptr);

n++;

return ptr;

}

void myfree(void \*ptr)

{

free(ptr);

printf("free(%p)\n", ptr);

m++;

}

void check()

{

if(m==n)

{

printf("There is no any memory leak\n");

}

if(n>m)

{

printf("you have memory leak\n");

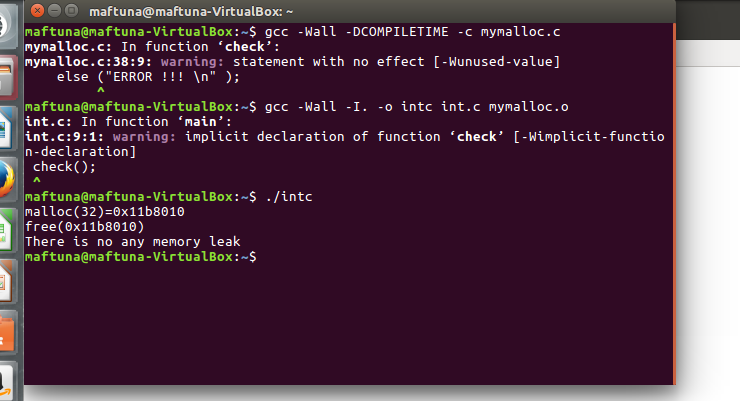
}

else ("ERROR !!! \n" );

}

#endif

Here shell for running above codes:



1. Codes for malloc and free functions to carry out inter-positioning at Link Time:

**int.c:**

#include <malloc.h>

#include <stdio.h>

int main()

{

int \*p=malloc(32);

free(p);

check();

return(0);

}

**Code for mymalloc.c:**

#ifdef LINKTIME

#include <stdio.h>

int n=0, m=0;

void\* \_\_realmalloc(size\_t size);

void \_\_real\_free(void \*ptr);

void \*\_\_wrap\_malloc(size\_t size)

{

void \*ptr=\_\_real\_malloc(size);

printf("malloc(%d)=%p\n", (int)size, ptr);

n++;

return ptr;

}

void \_\_wrap\_free(void \*ptr)

{

\_\_real\_free(ptr);

printf("free(%p) \n" , ptr);

m++;

}

void check()

{

if(m==n)

{

printf("There is no any memory leak! \n");

}

if(n>m)

{

printf("We have memory leak ! \n");

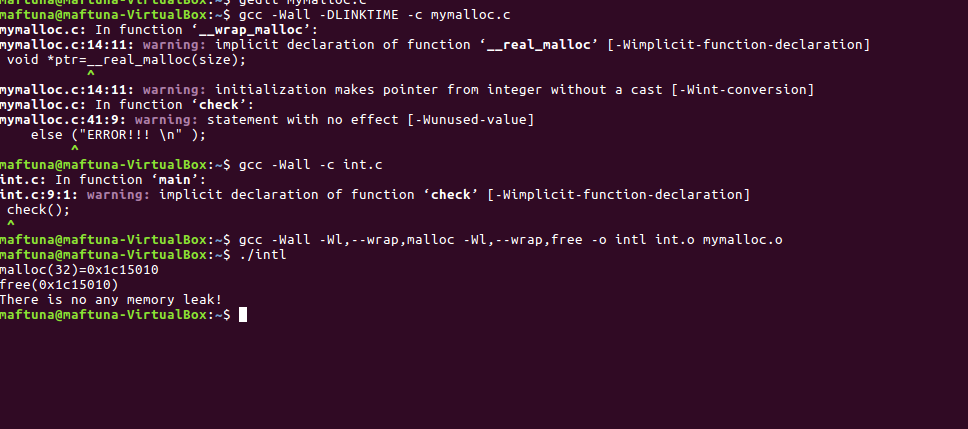
}

else ("ERROR!!! \n" );

}

#endif

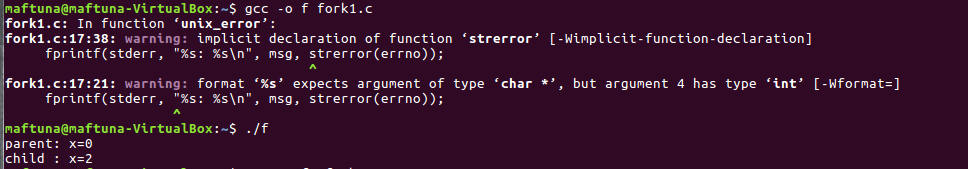
Here shell for running above codes:

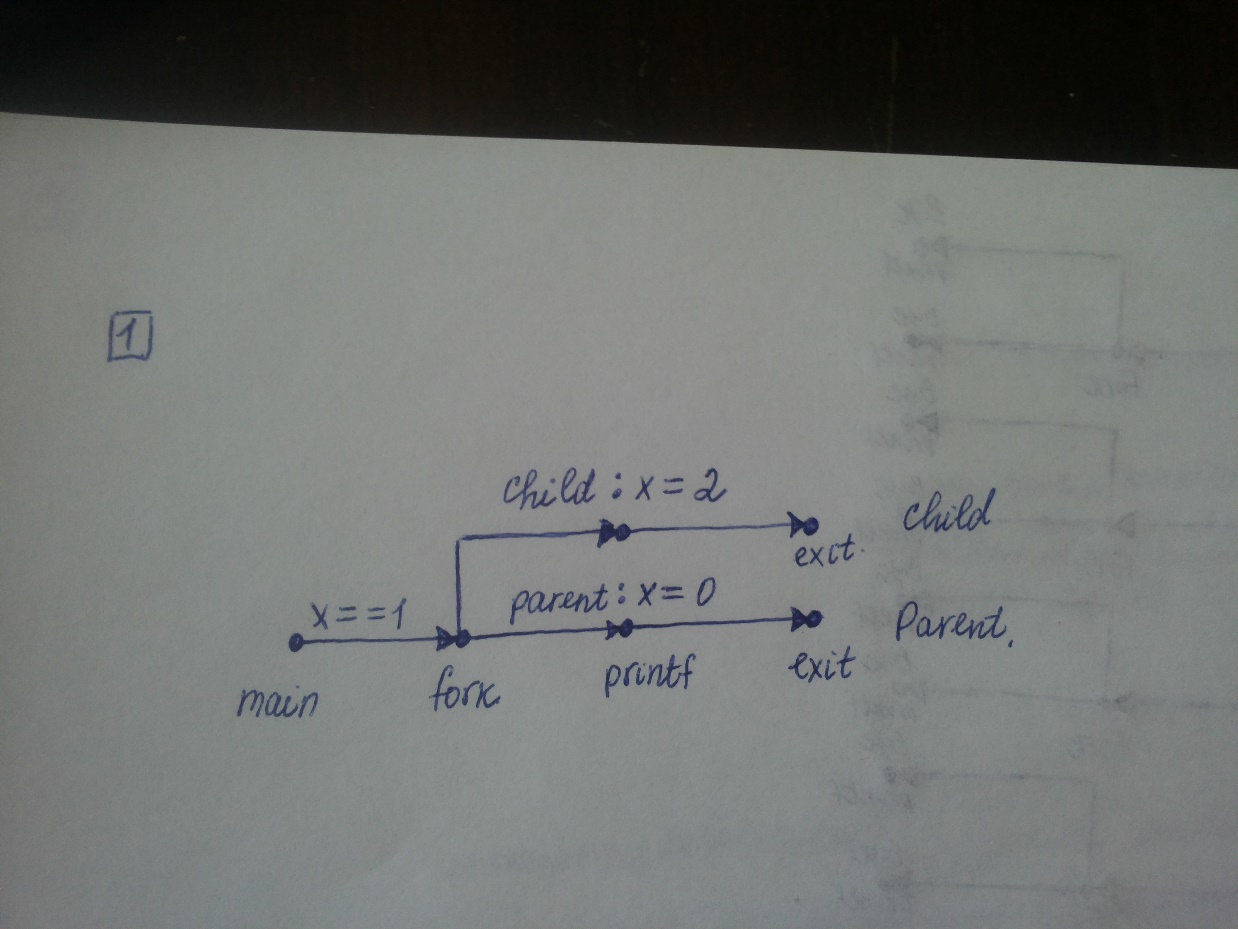


Answer for Question 3:

Here I provide all shells and graphs for each forks:

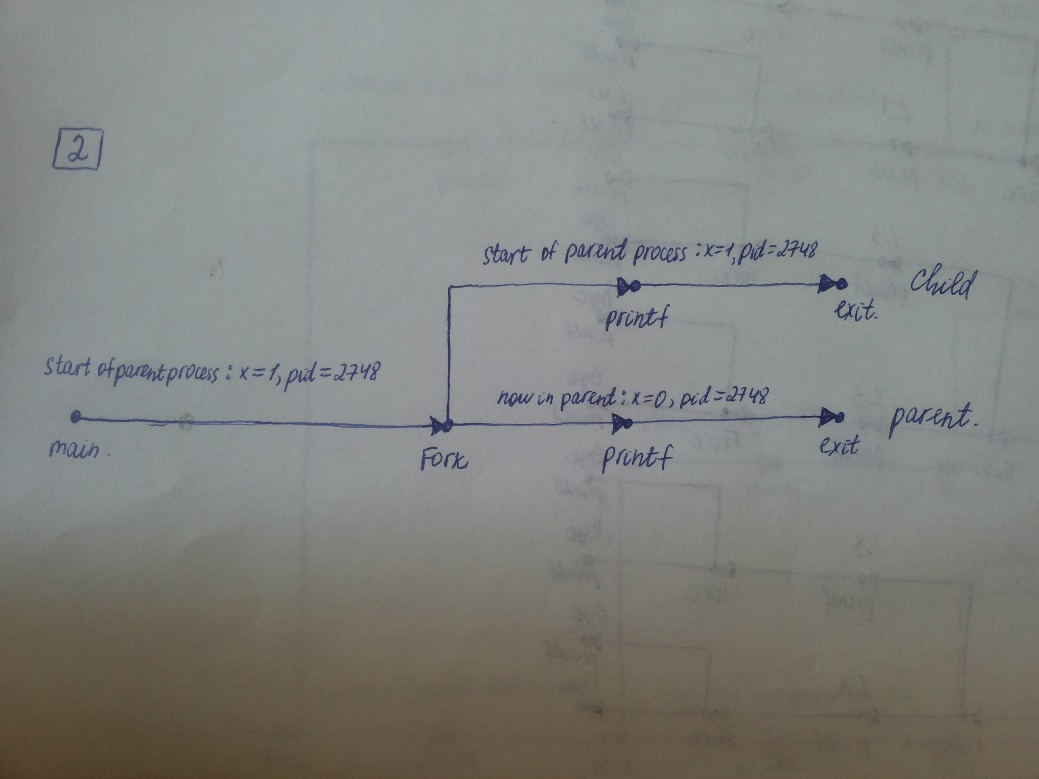
**For fork1():**

****

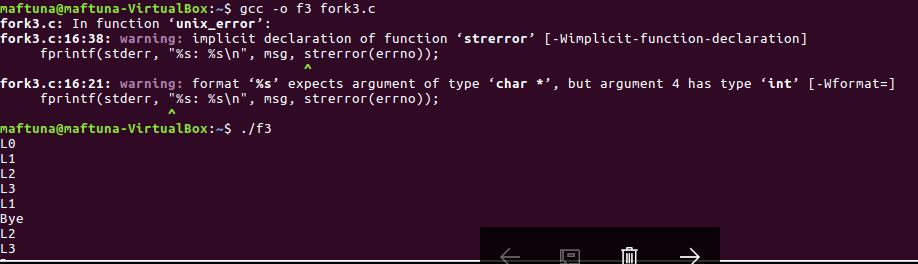
****

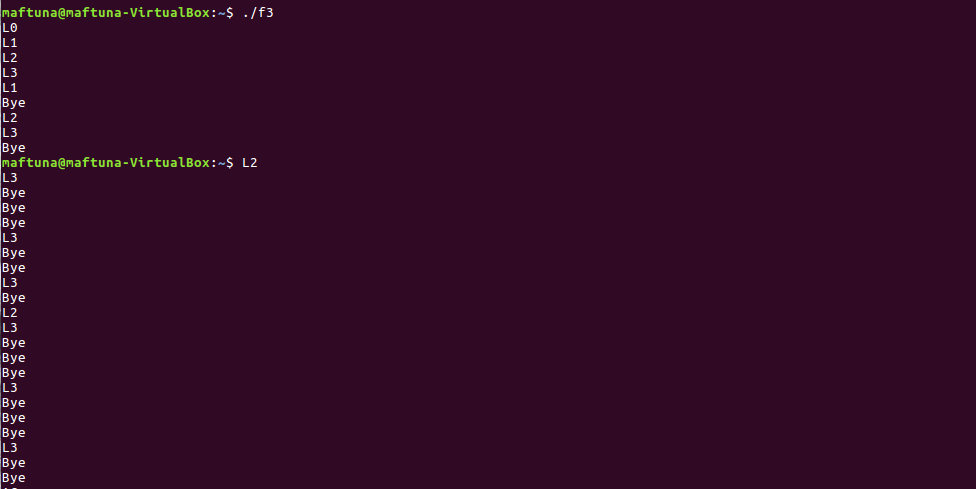
**For fork2():**

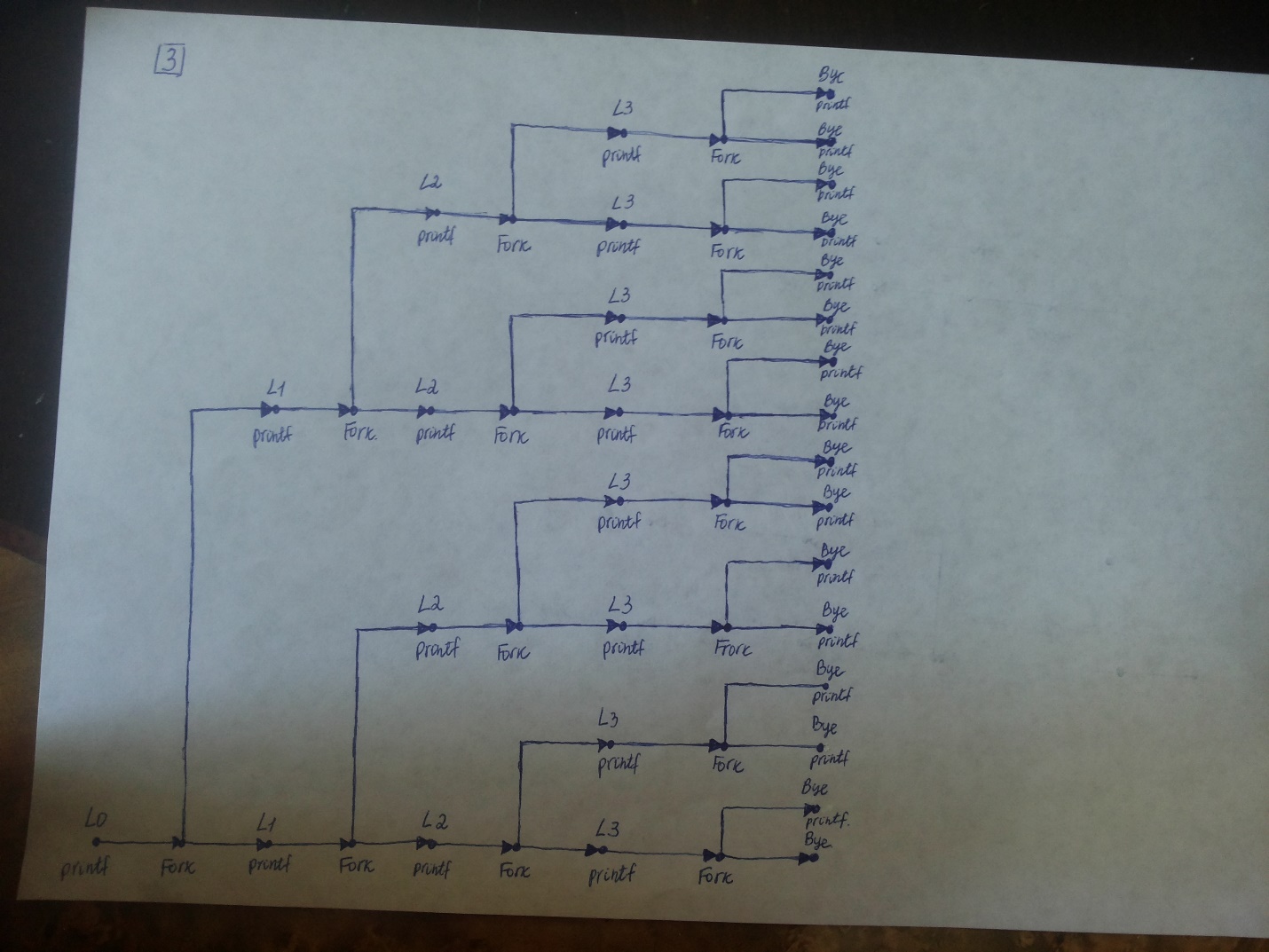




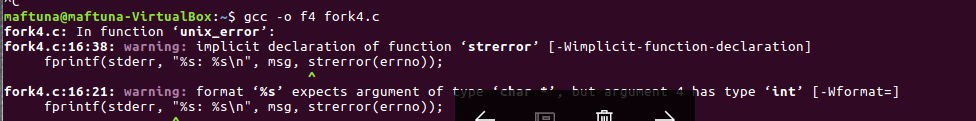
**For fork3():**

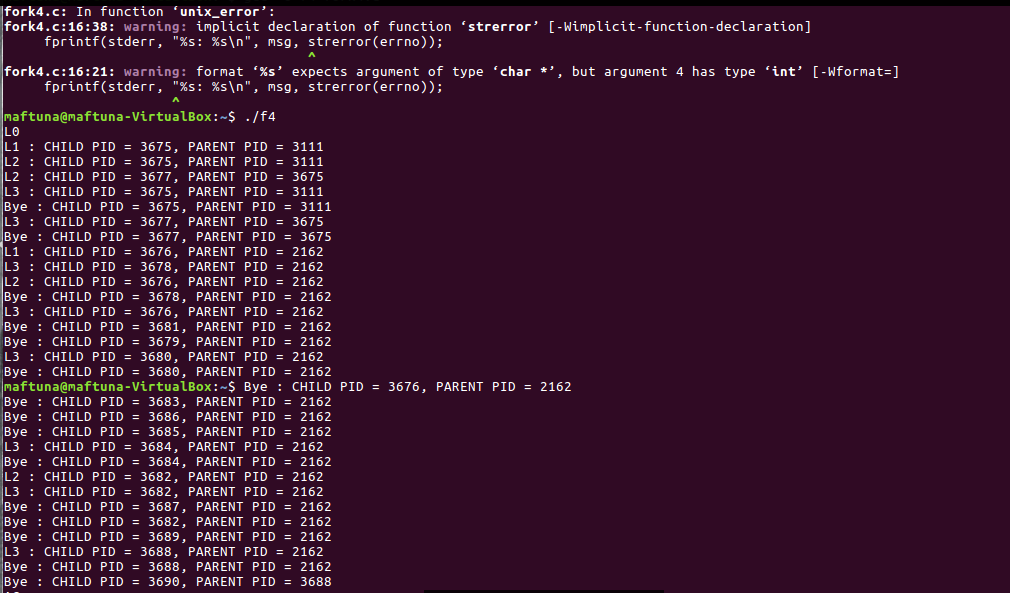
****

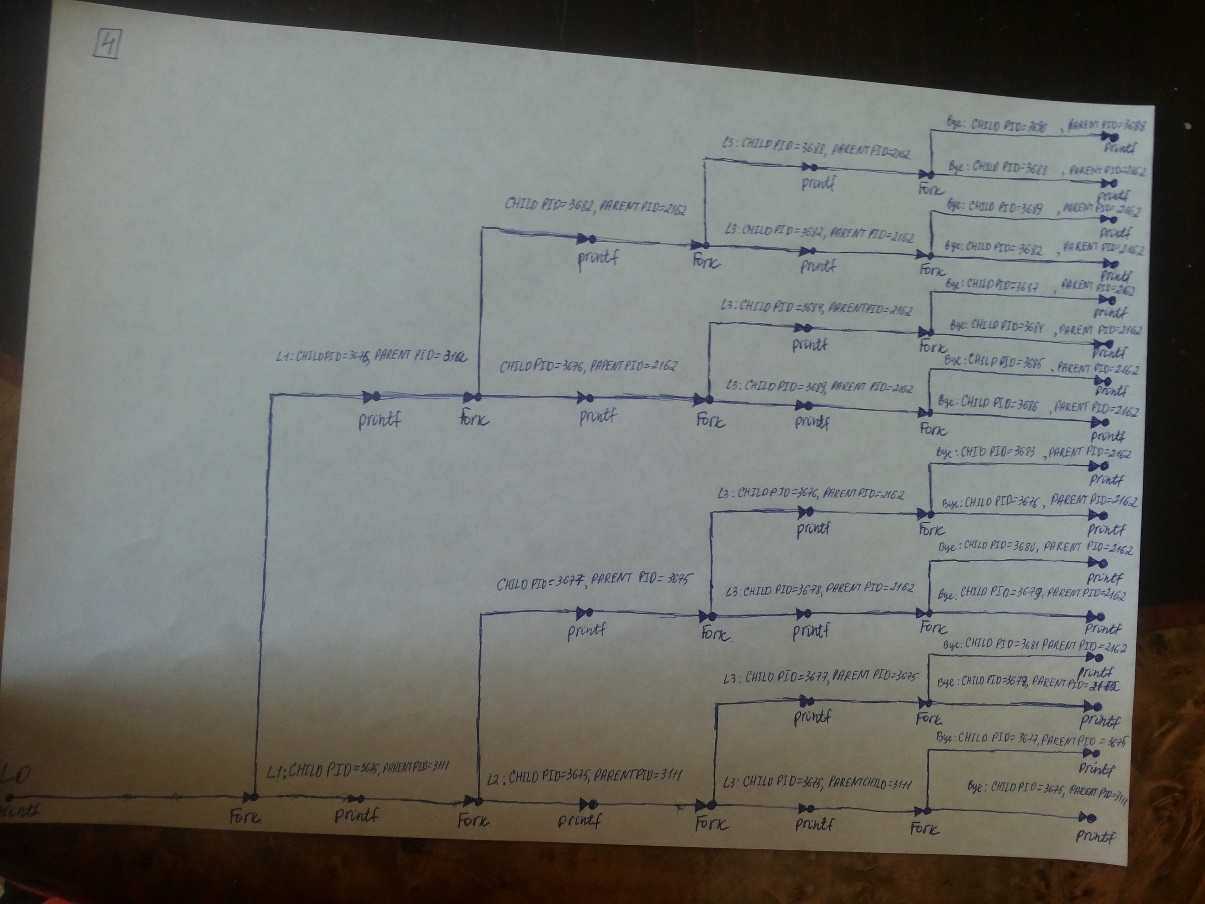
****

****

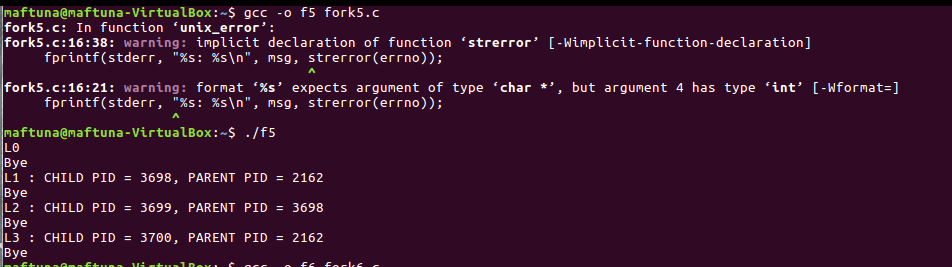
**For fork4():**

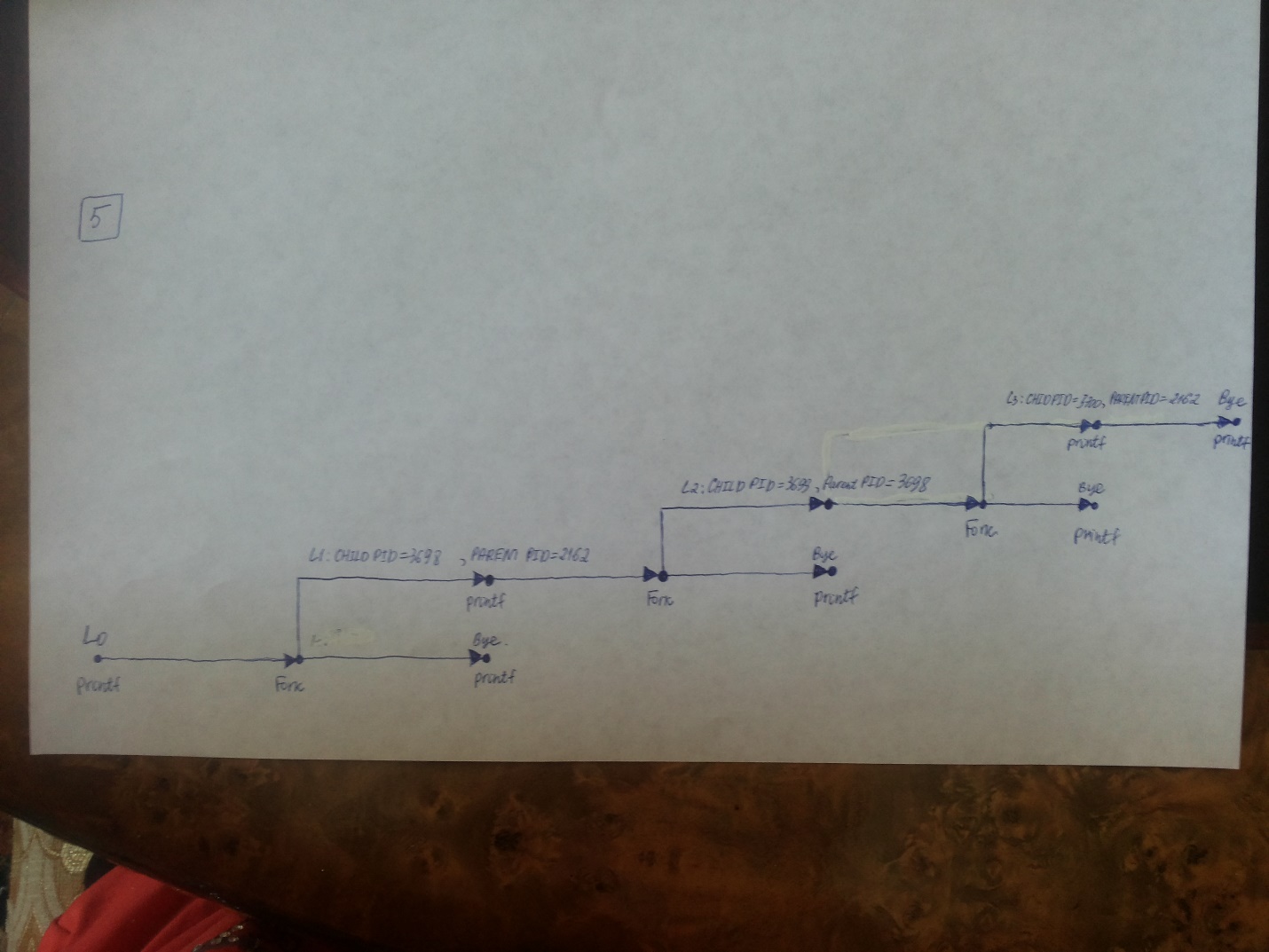
****

****

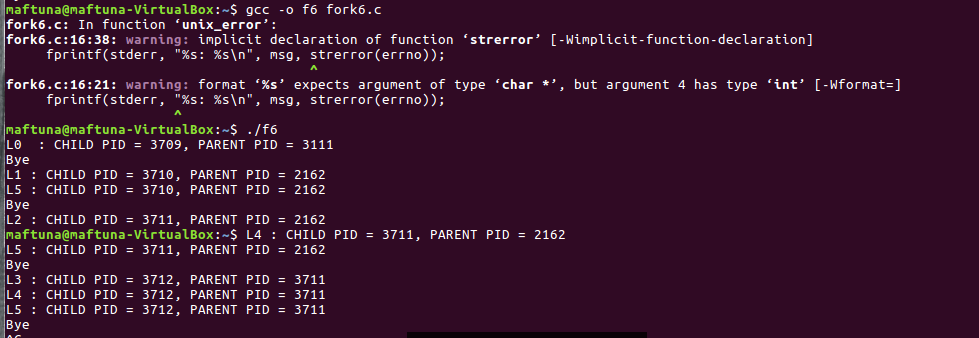
****

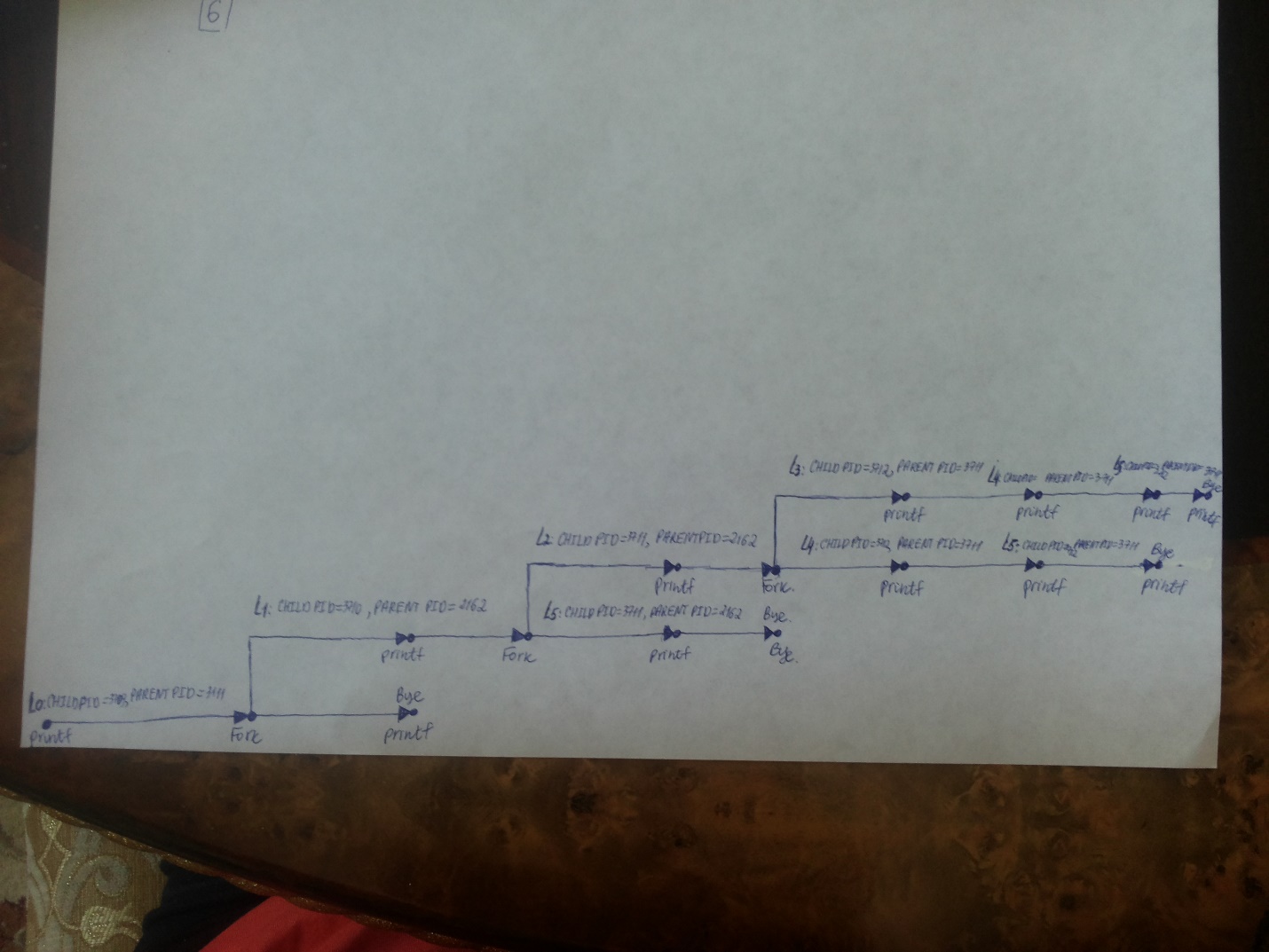
**For fork5():**

****

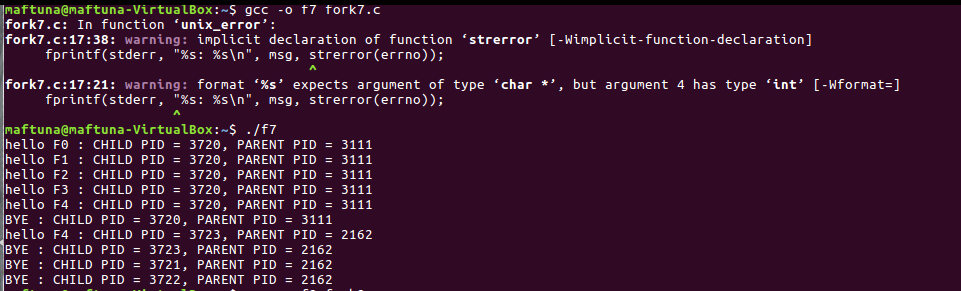
****

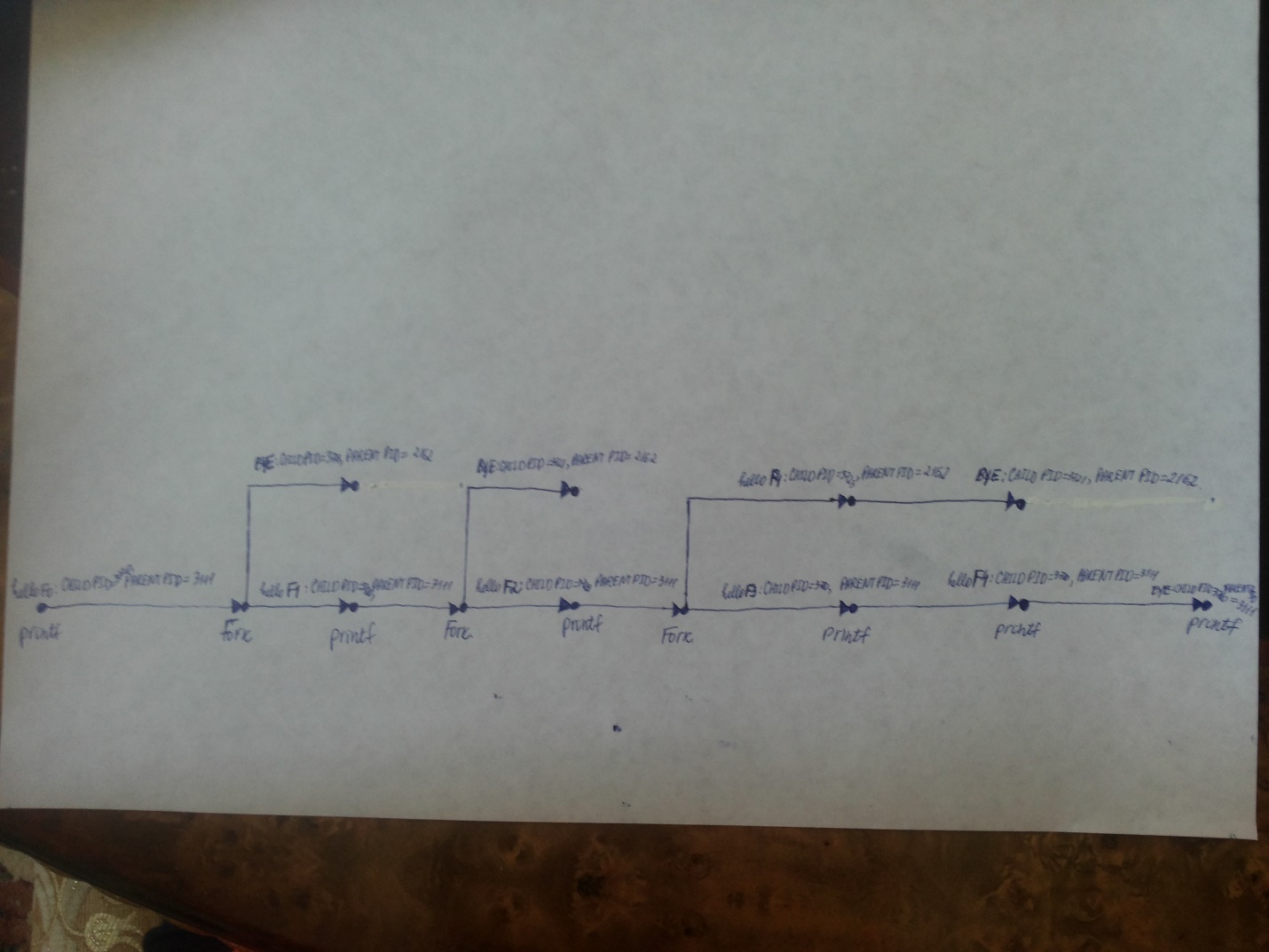
**For fork6():**

****

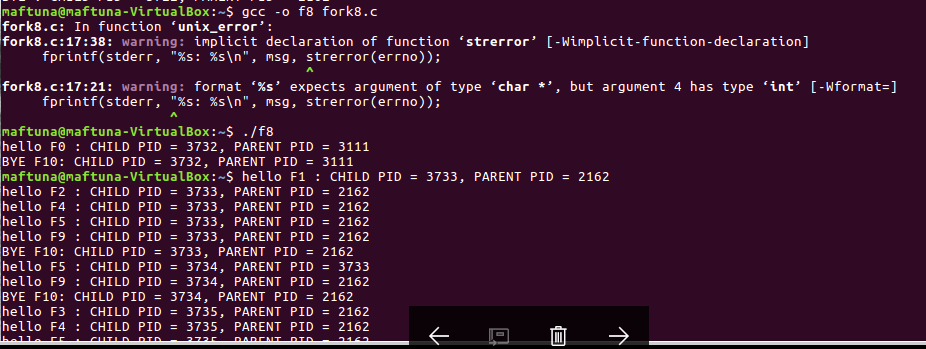
****

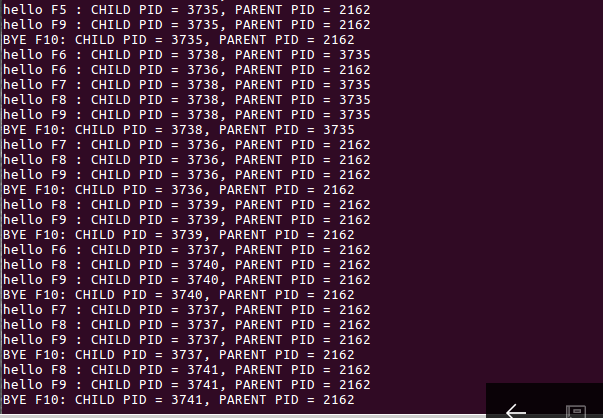
**For fork7():**

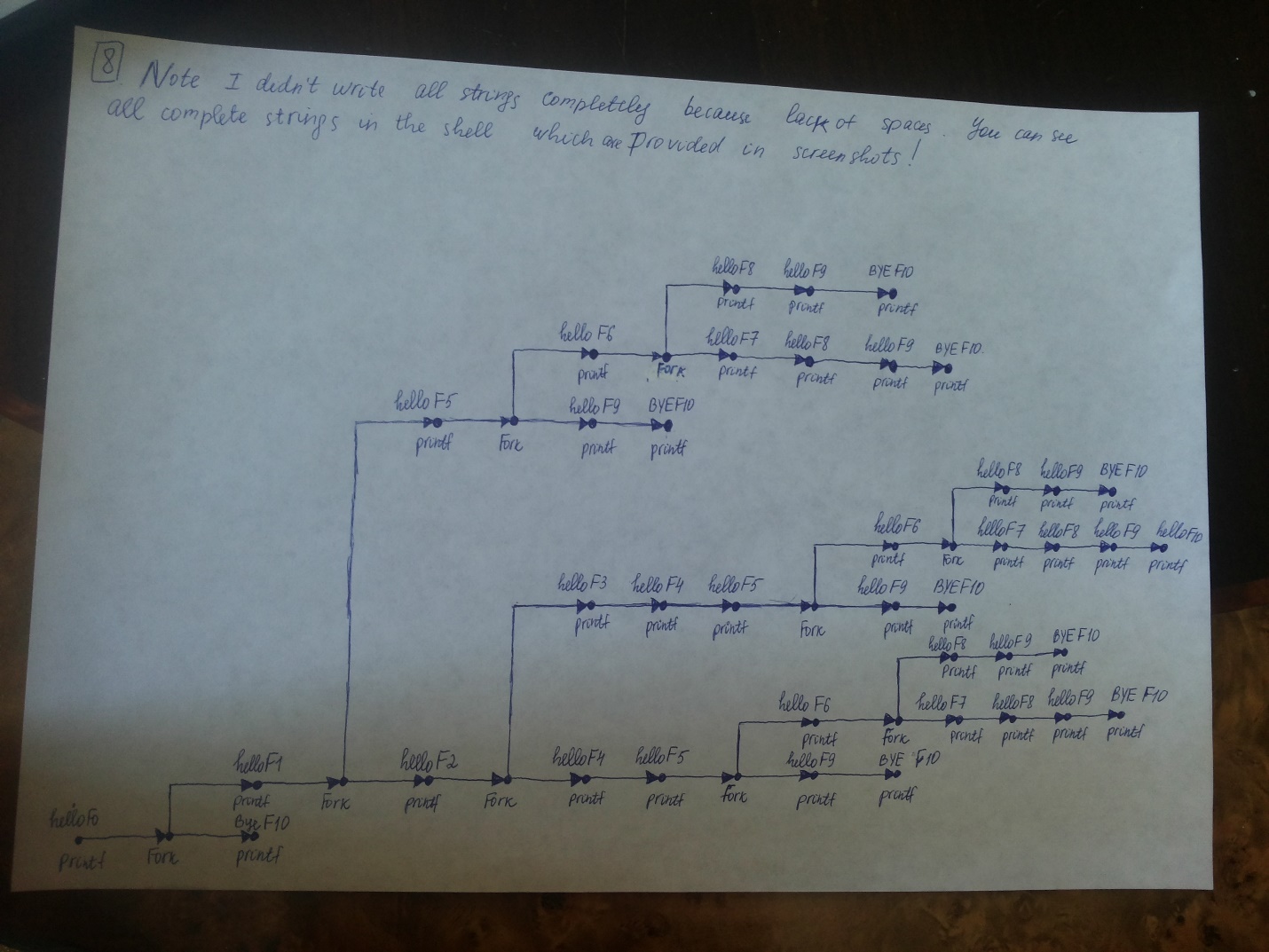
****



**For fork8():**

****

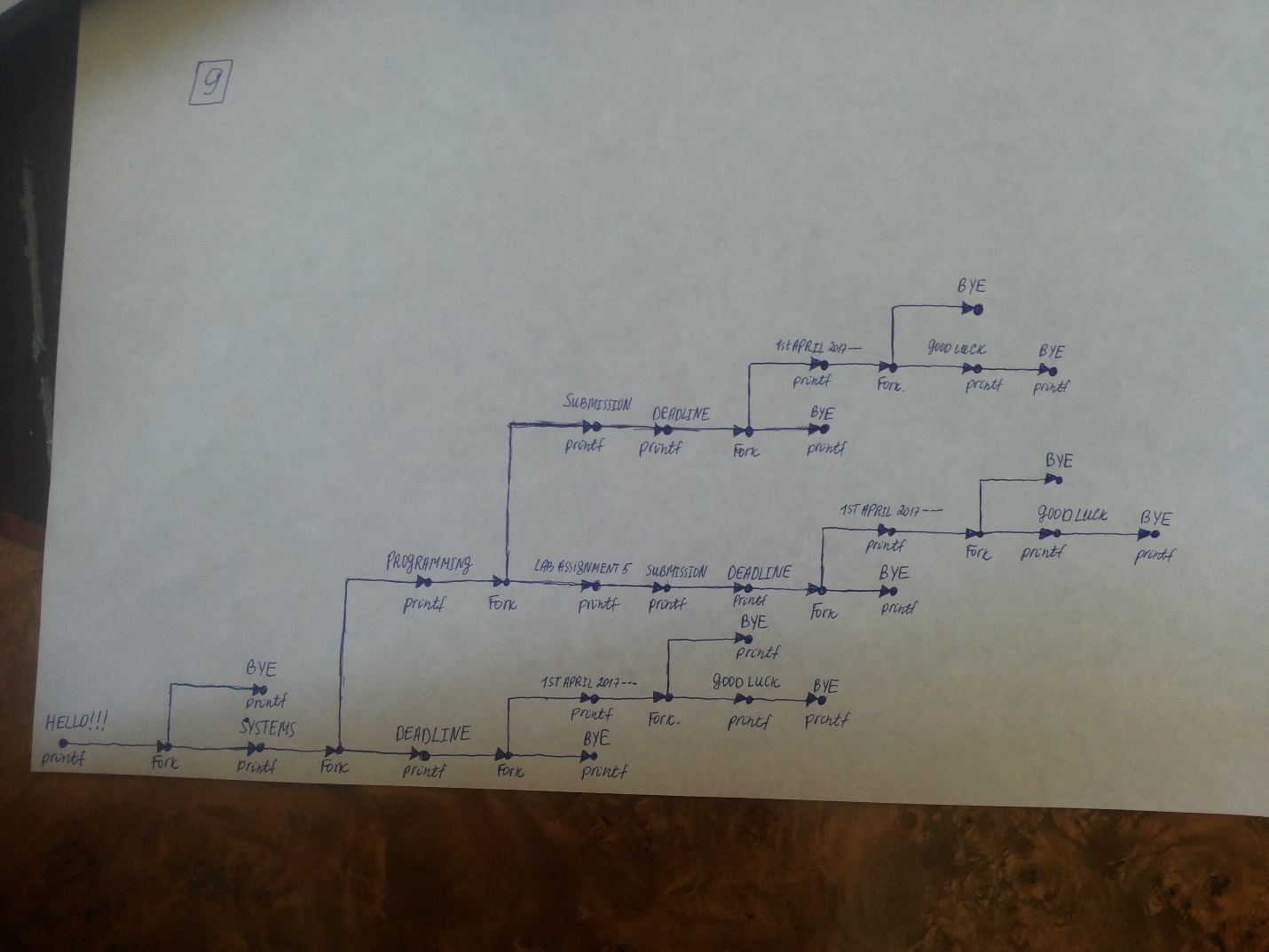
****

****

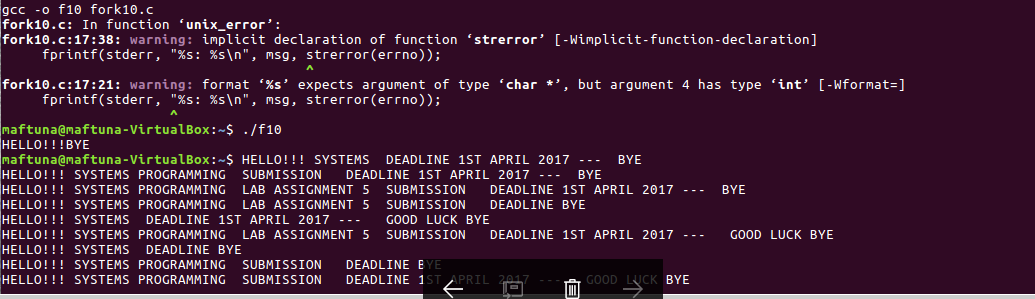
**Continues on next page ->>>**

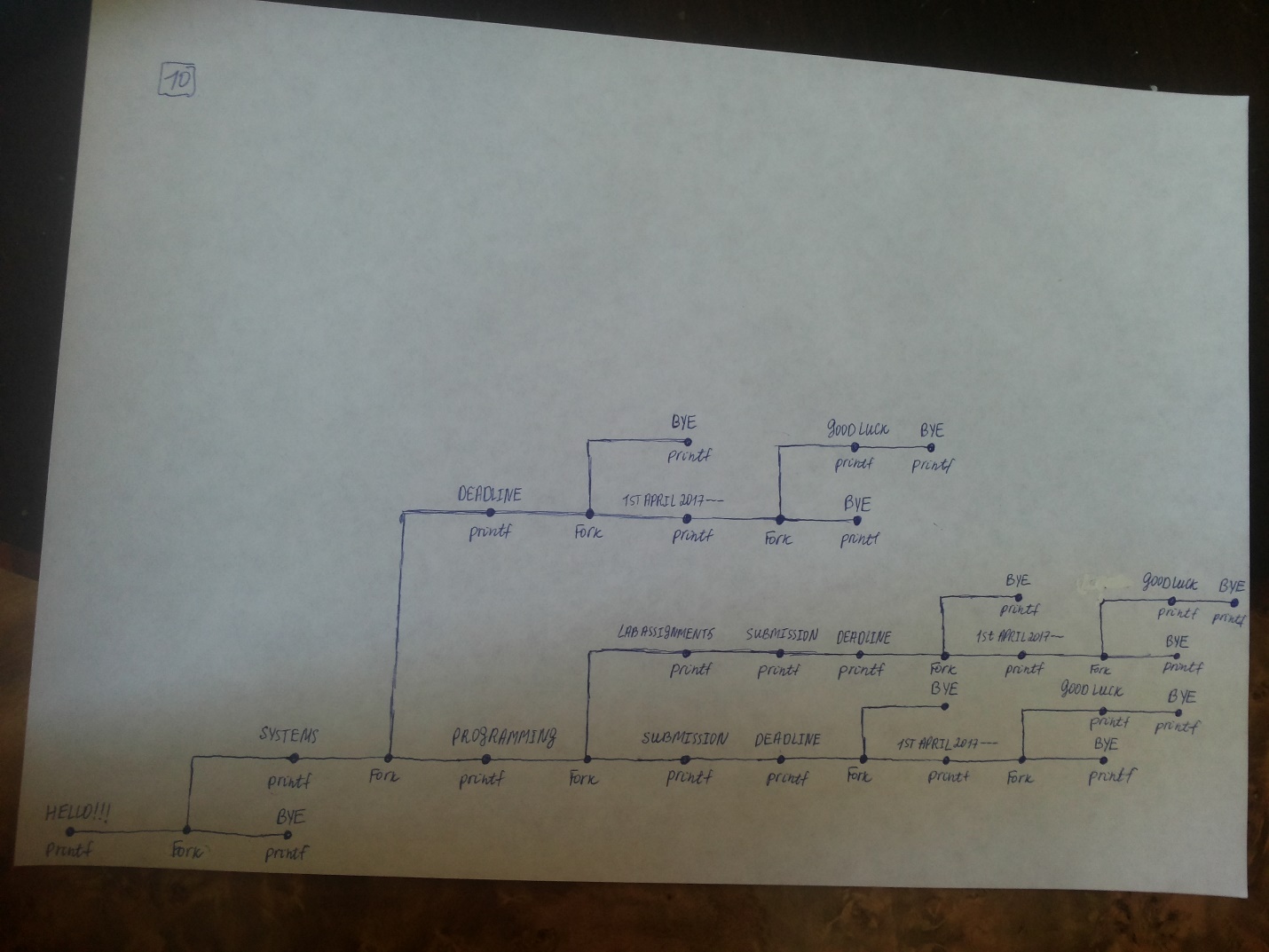
**For fork9():**

****

****

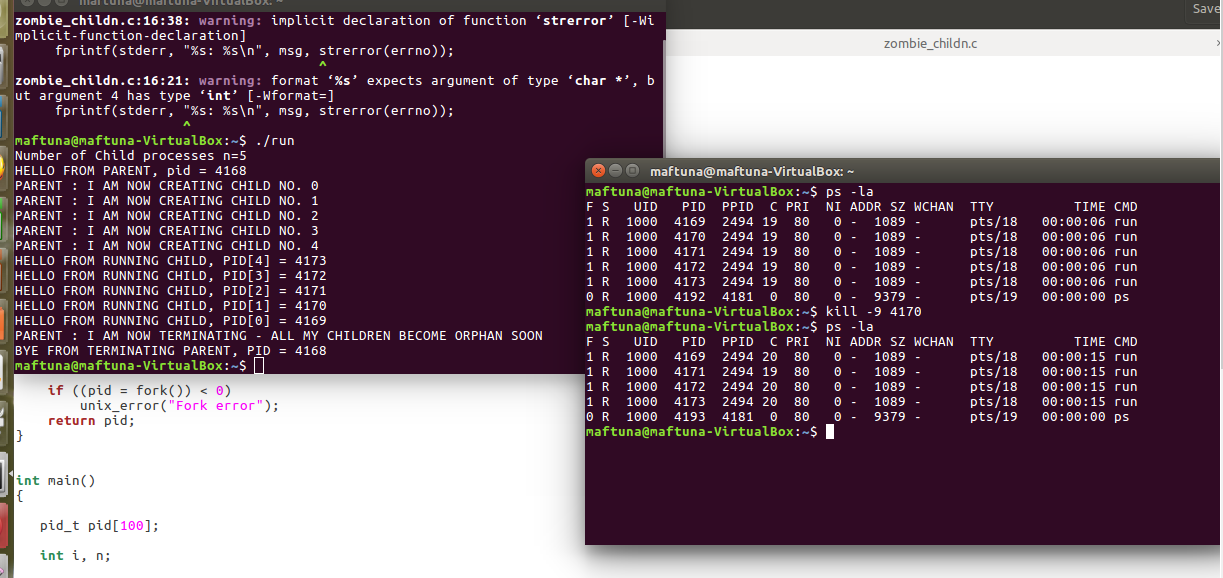
**For fork10():**

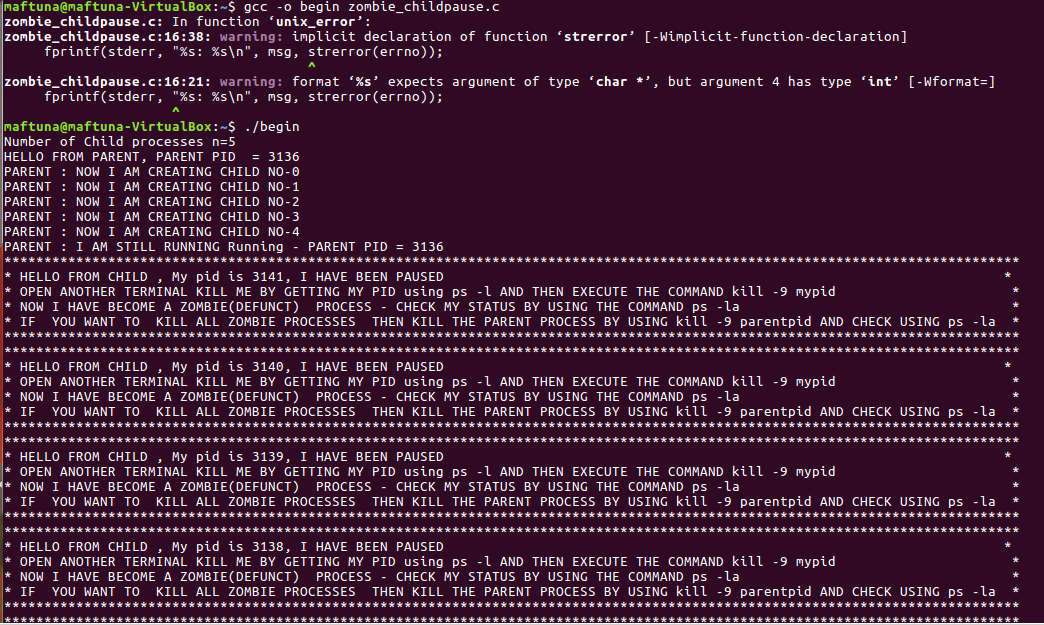
****

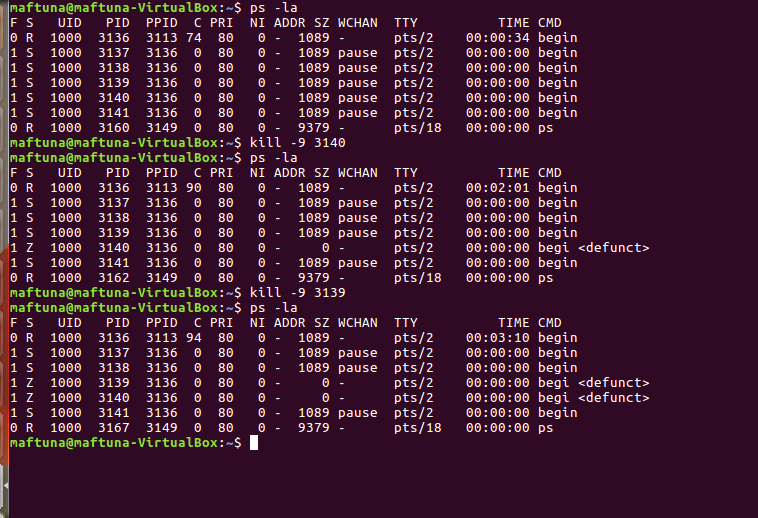
****

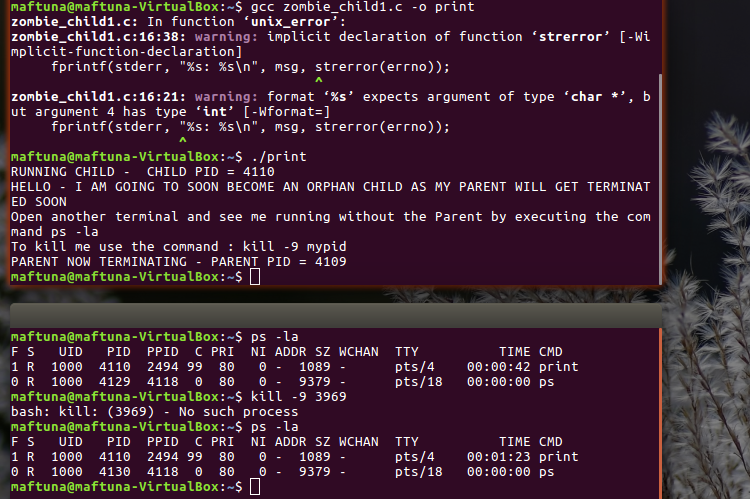
**Answer for question 4:**

To display orphan child we use ps –la command. By kill -9 mypid we can kill child. Inscreenshot following, you can see in first shellcomplete data and on another terminal changes after using kill -9 command:

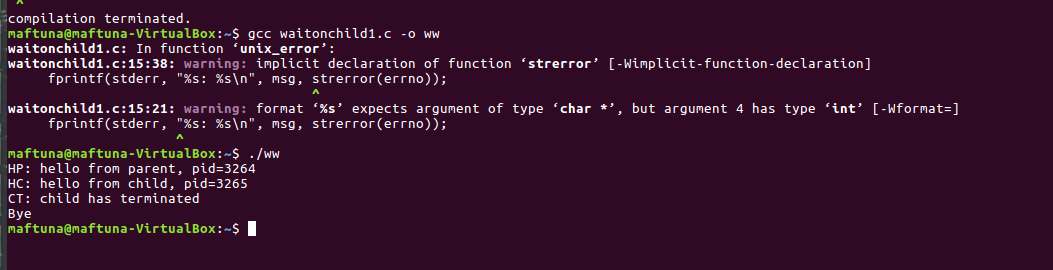


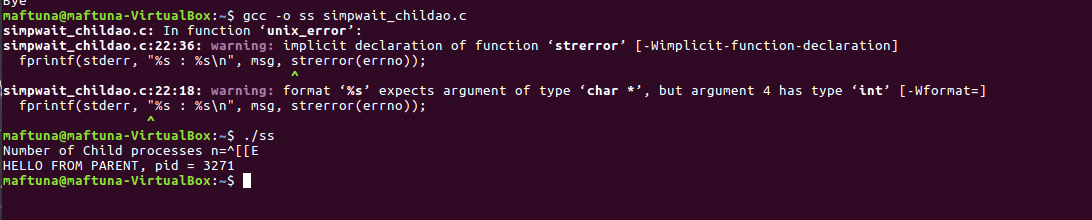


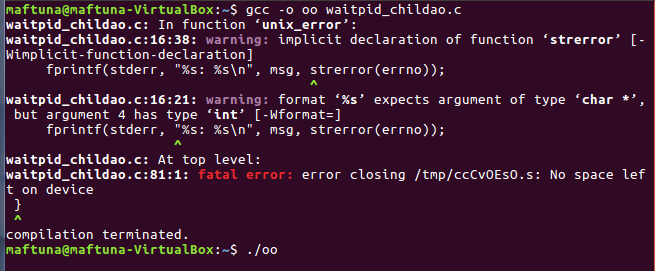




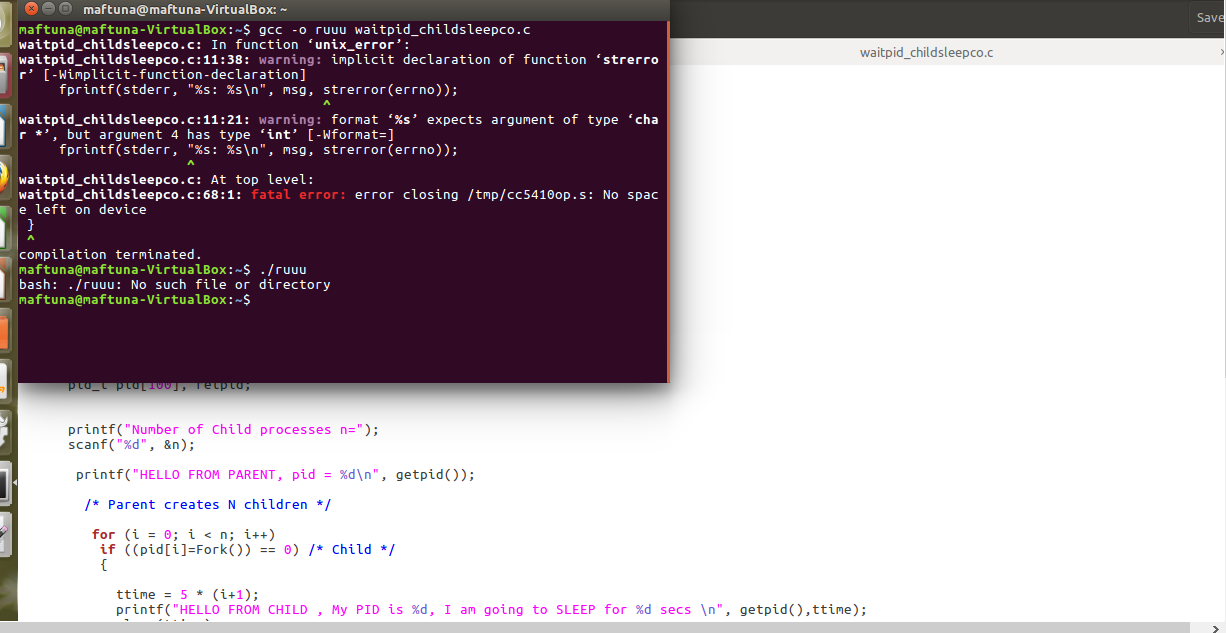
**Answer for question 5:**

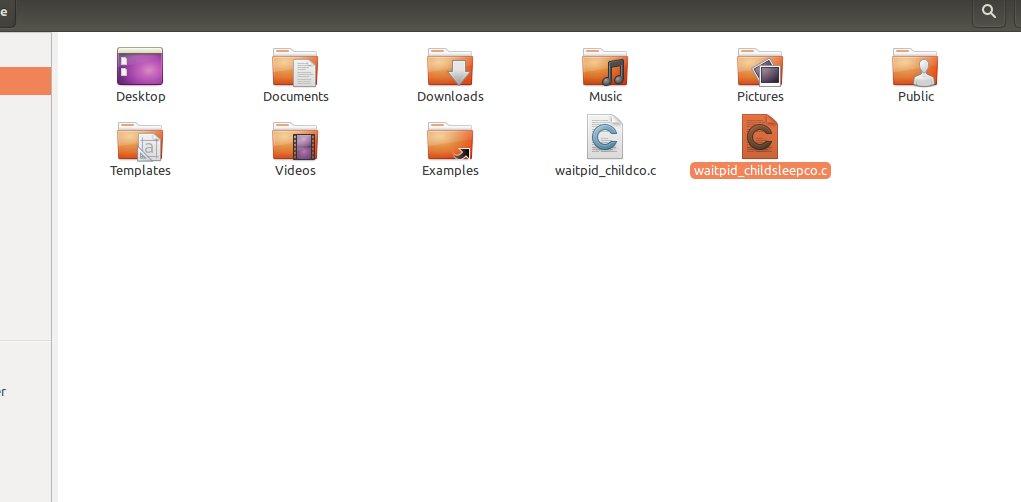




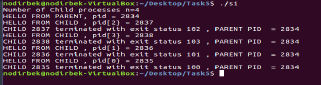


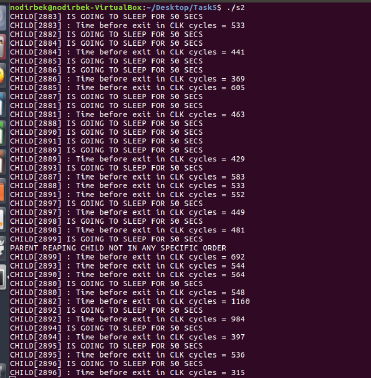
I could not run these codes in my linux and I had not time to fix it becaouse of deadline and there remained less time to prepare final exams. However I used my friend’s linux and here what I did. But firsly I want to give you evidence about problem with my linux:

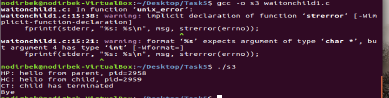


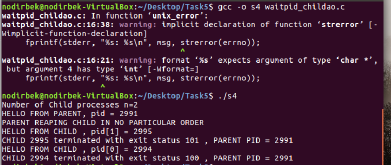


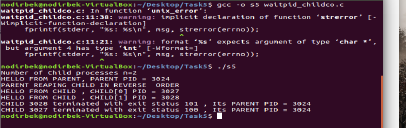
Answer continues:

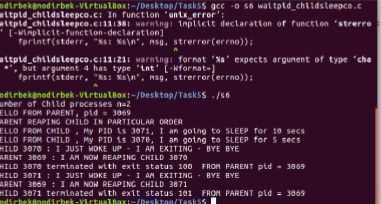












Here I want to give some comments about codes. In code of waitonchild1.c parent waits some childs to terminate. Before creating next child every child is built then should be terminated. In code of simpwait\_childao.c parent reaps child orderly . in code of waitpid\_childao.c parent reaps children reversly. In code of waitpid\_childco.c children come and sleep then parent reaps all sleepy children. Thanks for your understanding the last problem with my linux and I wish have a nice summer!!! ☺