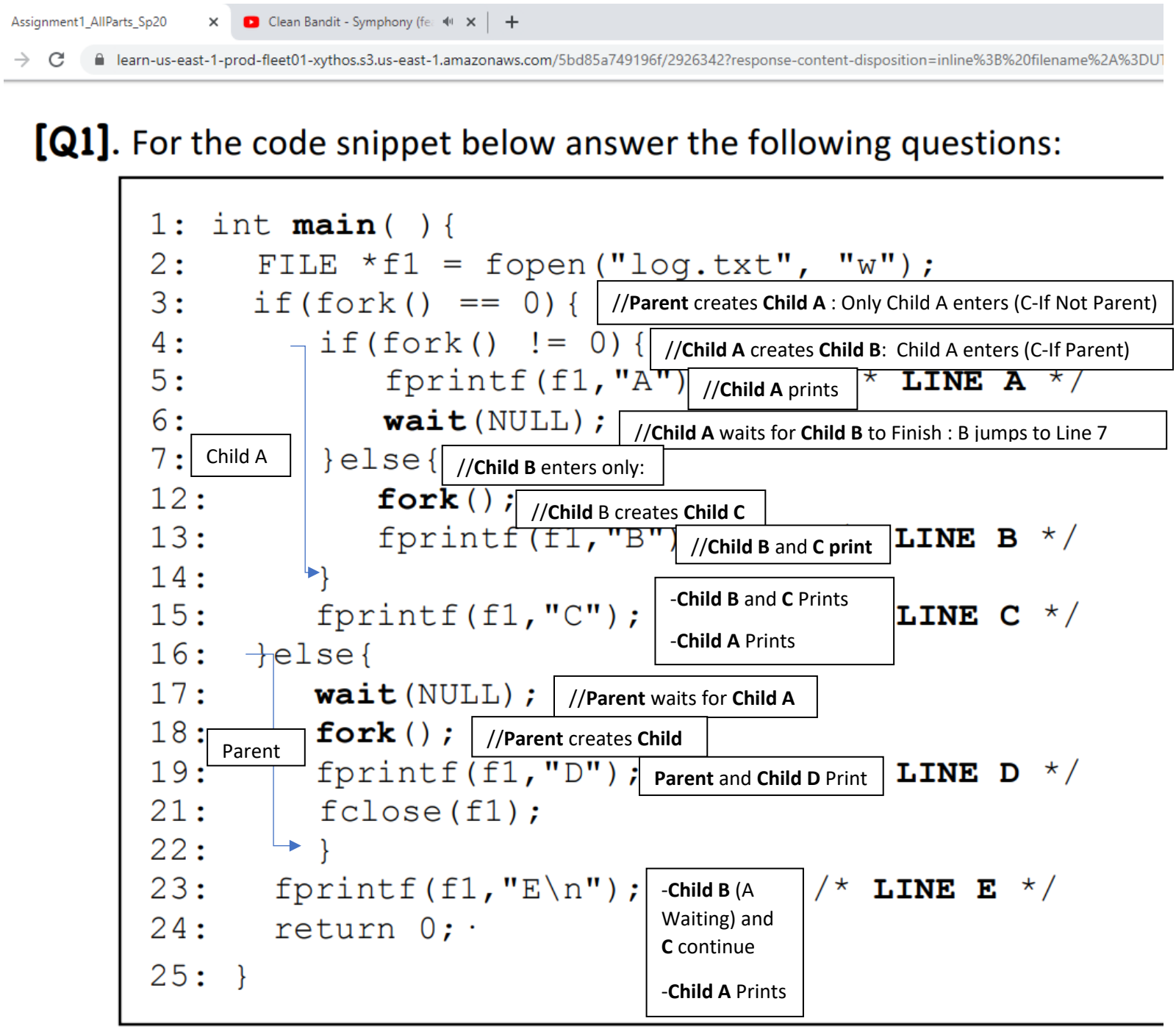
Agnes Duru 2/28/2020

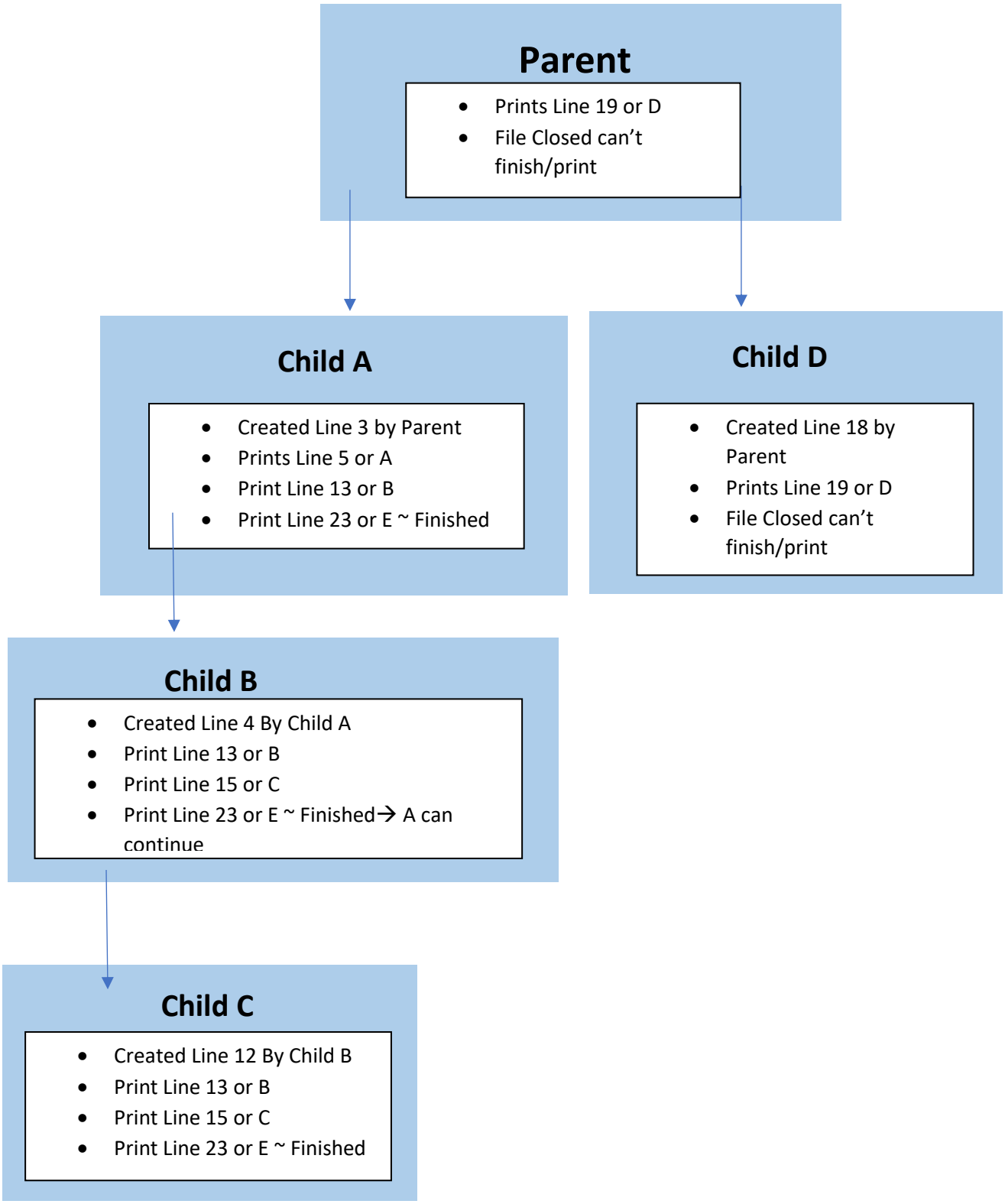
Operating Systems Assignment 1

Written Part:

- Q1] → Question #1 :



A.


Parent
Prints Line 19 or D
File Closed can't finish/print
Child A
Created Line 3 by Parent
Prints Line 5 or A
Print Line 13 or B
Print Line 23 or E ~ Finished
Child D
Created Line 18 by Parent
Prints Line 19 or D
File Closed can't finish/print
Child B
Created Line 4 By Child A
Print Line 13 or B
Print Line 15 or C
Print Line 23 or E ~ Finished→ A can continue
Child C
Created Line 12 By Child B
Print Line 13 or B
Print Line 15 or C
Print Line 23 or E ~ Finished

A.

B.

Output:

- Child A prints : “A” (Line 5 or A)
- Child B prints: “B” (Line 13 or B)
- Child C prints: “B” (Line 13 or B)
- Child B print “C” (Line 15 or C)
- Child C prints “C” (Line 15 or C)
- Child B prints “E\n” (Line 23 or E)
- Child C prints “E\n” (Line 23 or E)
- Child A prints: “B” (Line 13 or B)
- Child A print “E\n” (Line 23 or E)
- Parent prints: “D” (Line 19 or D)
- Child D prints “D” (Line 19 or D)

  s

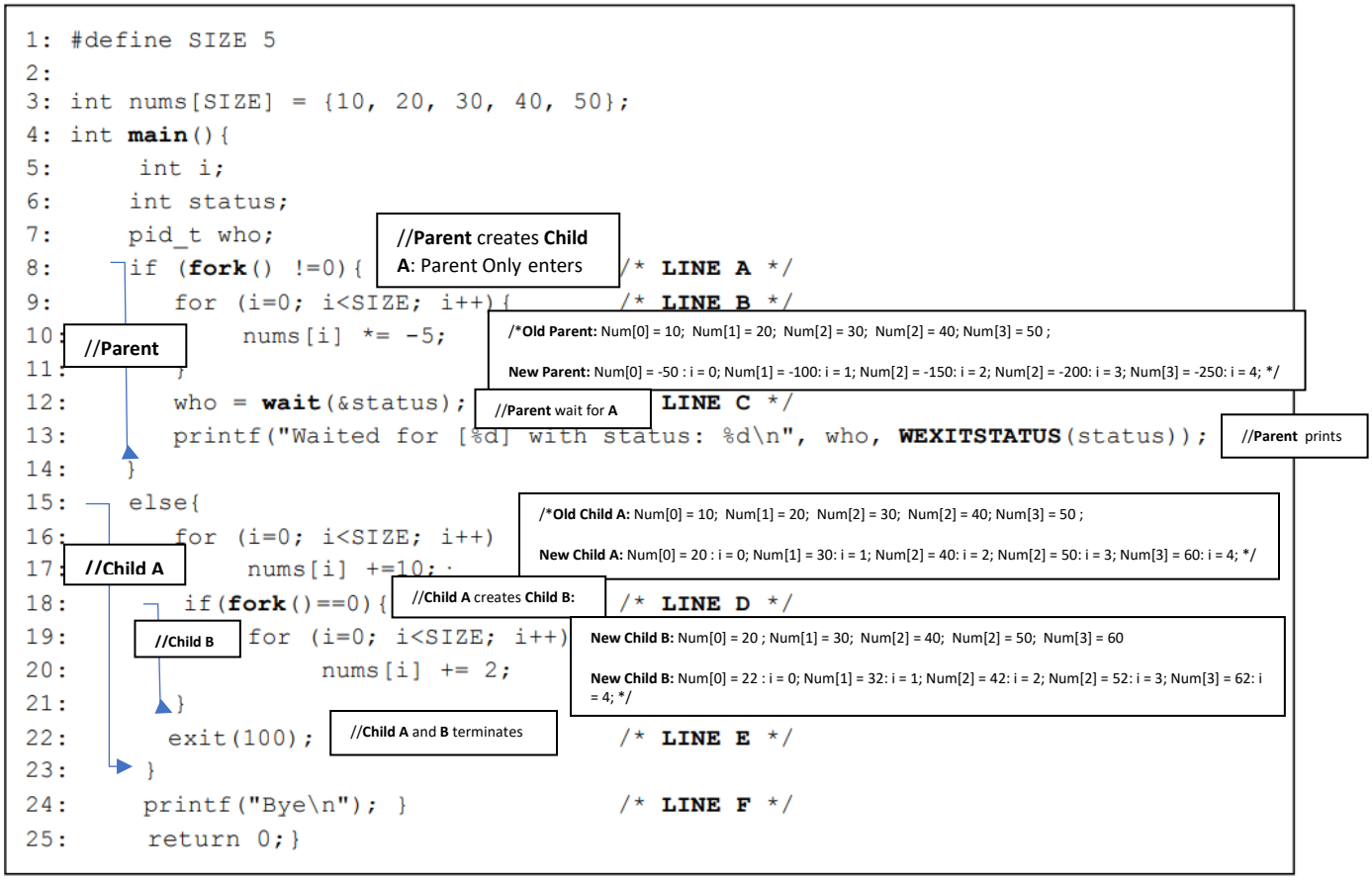
Questions:

B. File “log.txt” is opened once (line 2), but all processes can write data to it! Briefly explain why/how this is possible.
   - When fork() is called the Parent processes creates a Child processes that inherits a lot of the Parent’s open-file table. This gives the Child process the ability to also write data to the opened file.

C. What does line 21 do? Briefly explain your answer. (Note: the obvious answer that it closes a file is not sufficient!). Hint: Assume you are commenting out line 21, what will change? Justify your answer.
   - Line 21 stops the Parent and it’s child -Child D from having access to the file and so they cannot execute the printf statement. Removing that line would enable the Parent and Child D to both print “E\n”.

- [Q2] →Question 2

[Q2]. For the code snippet below answer the following questions:



A.

Parent

- [Line 3]: **Old Parent:** Num[0] = 10; Num[1] = 20; Num[2] = 30; Num[2] = 40; Num[3] = 50 ;
- [Line 10]: **New Parent:** Num[0] = -50 : i = 0; Num[1] = -100: i = 1; Num[2] = -150: i = 2; Num[2] = -200: i = 3; Num[3] = -250: i = 4;
- [Line 13] **Parent** Prints
- [Line 24 or F] **Parent** Prints

Child A

- Created Line 8 by **Parent**
- [Line 3] : **Old Child A:** Num[0] = 10; Num[1] = 20; Num[2] = 30; Num[2] = 40; Num[3] = 50 ;
- [Line 17] : **New Child A:** Num[0] = 20 : i = 0; Num[1] = 30: i = 1; Num[2] = 40: i = 2; Num[2] = 50: i = 3; Num[3] = 60: i = 4;
- [Line 22 or E] → **Child A** terminates does **NOT** execute [Line 24]

Child B

- Created Line 18 by **Child A**
- [Line 3] : **Old Child B:** Num[0] = 20; Num[1] = 30; Num[2] = 40; Num[2] = 50; Num[3] = 60 ;
- [Line 20] : **New Child B:** Num[0] = 22 : i = 0; Num[1] = 32: i = 1; Num[2] = 42: i = 2; Num[2] = 52: i = 3; Num[3] = 62s: i = 4;
- [Line 22 or Es] → **Child B** terminates does **NOT** execute [Line 24]

B. Output :

- Parent prints [Child A] "waited for 2001 for status 100" or [Child A] "waited for 2002 for status 100" (Line 13).
- Parent prints "bye" (Line 24 or F).

Questions:

B. How many copies of the original variable nums are there? For each one of these copies, mention their initial and final values.

- There are three copies of the variable nums.
- The Parent process :
  - **Old Parent:** Num[0] = 10; Num[1] = 20; Num[2] = 30; Num[2] = 40; Num[3] = 50 ;
  - **New Parent:** Num[0] = -50 : i = 0; Num[1] = -100: i = 1; Num[2] = -150: i = 2; Num[2] = -200: i = 3; Num[3] = -250: i = 4;
- The Child A process:
  - **Old Child A:** Num[0] = 10; Num[1] = 20; Num[2] = 30; Num[2] = 40; Num[3] = 50 ;
  - **New Child A:** Num[0] = 20 : i = 0; Num[1] = 30: i = 1; Num[2] = 40: i = 2; Num[2] = 50: i = 3; Num[3] = 60: i = 4;
- The Child B process:
  - **Old Child B:** Num[0] = 20; Num[1] = 30; Num[2] = 40; Num[2] = 50; Num[3] = 60 ;
  - **New Child B:** Num[0] = 22 : i = 0; Num[1] = 32: i = 1; Num[2] = 42: i = 2; Num[2] = 52: i = 3; Num[3] = 62: i = 4;

C. Assuming that the parent process' pid is 2000, and its children processes's pids are the successive numbers 2001, 2002, … explain what could be the possible value(s) that variable who can have (in Line C).

- The only possible values are 2001 for Child A and 2002 Child B. This is because the wait() function allows any child process to finish first and so child A could have finished first and returned the value 2001 or B could have finished first and returned 2002.

D. WFEXITSTATUS is a macro used to present the value of status as an integer. What will be the value of status in Line C? Briefly explain your answer.

- The value of status will be 100 because 100 is what will get returned by the Exit method.

E. Indicate which processes (from the ones identifies in (a)) are going to execute the printf-statements in lines 13 and 24. Briefly justify your answer.

- The parent will execute lines 13 and 24/ A. The parent will execute line 13 because of the condition on line 8/A. Only the parent can enter that if statement because the parent is greater than 0 and the child is equal to 0. The parent will also execute line

24 because both Child A and Child B get terminated on line 22 and thus cannot execute line 22s.

F. Identify the memory segment (stack-, heap- or data segment) in which each variable of this program resides. Note since you have more than one processes describe these three memory segments for each one of these processes (unless they are identical)
   - The variable ‘nums’ is part of the data segment since it is a global variable.
   - The variable ‘i’ is part of the stack segment since it is a local variable.
   - The variable ‘status’ is part of the stack segment since it is a local variable.
   - The variable ‘who’ is part of the stack segment since it is a local variable.

[Q3]. For the code snippet below answer the following questions:

```
1: int main(int argc, char **argv){
2:
3:     char *c1_argv[]={"/usr/bin/ncal",
4:                      "-e",
5:
6:                                    ;
7:     pid_t pid = fork();
8:     if(pid == 0){
9:             execv(argv[0], c1_argv);      /* LINE A */
10:            printf("Aloha");
11:    }else{
12:            waitpid(pid, NULL
13:
14:            ork() == 0){
15:                    printf("Hello
16:                    while(1) {;}
17:            }else{
18:                    printf(f1, "Ciao\n");
19:            }
20:    }
21:    printf("Bye\n");                  /* LINE D */
22:    return 0;
23: }
```

//Parent creates Child A

//Child A enters

//Child

//Child A now only executes function and terminates

//Child A does not Execute this line

//Parent enters

//Parents doesn't have a child to wait for; Continues

//Parent creates Child B; Child B enters

//Parent

//Child B

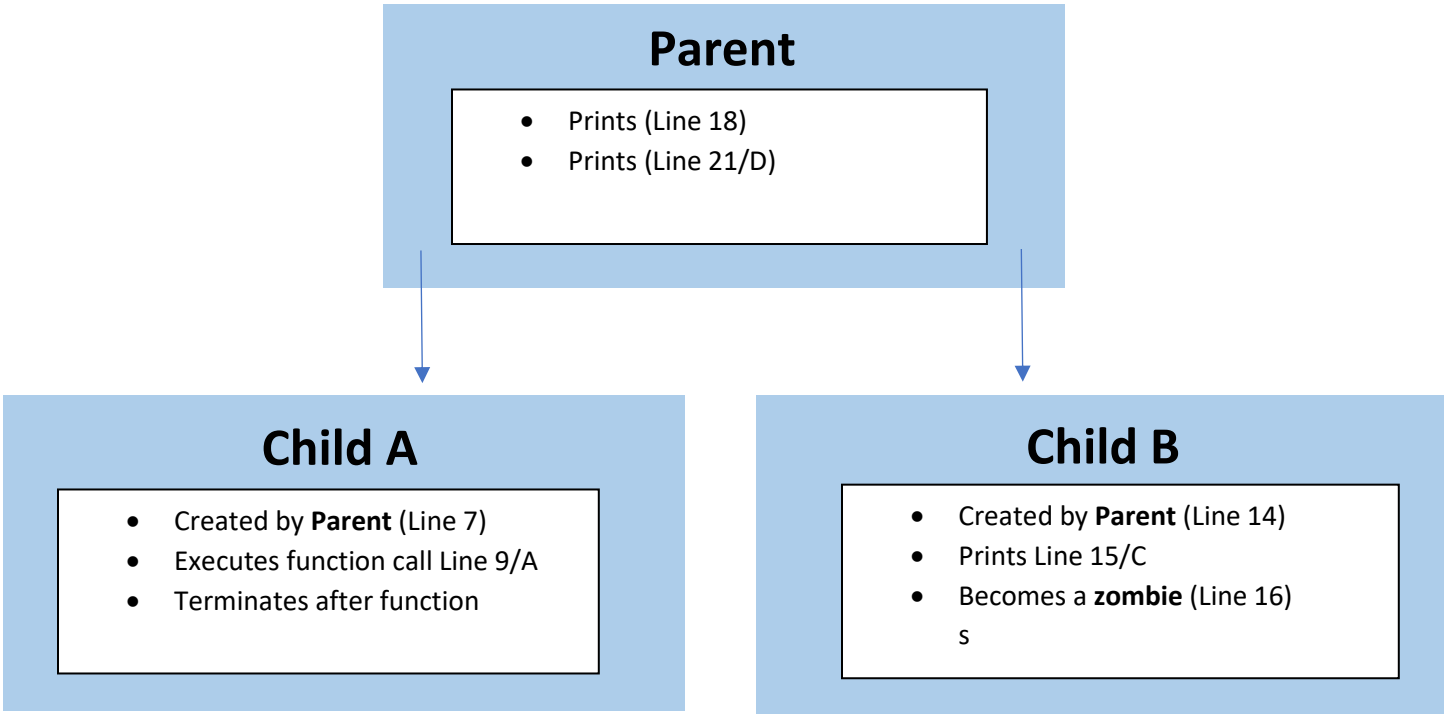
//Child B Prints

//Child B stuck in while loop and becomes a zombie

//Parent Prints

//Parent sPrint

B.


Parent
Prints (Line 18)
Prints (Line 21/D)
Child A
Created by **Parent** (Line 7)
Executes function call Line 9/A
Terminates after function
Child B
Created by **Parent** (Line 14)
Prints Line 15/C
Becomes a **zombie** (Line 16)
s

Output:

- **Child B** prints “Hello” (Line 15/C)
- **Parent** prints “Ciao” (Line 18)
- **Parent** prints “Bye” (Line 21/D)

Questions:

B. How many times is the string “Bye” (LINE D) getting printed? Briefly justify.

- The string Bye is getting printed once. It gets printed by the Parent. Child A does not print it because it continued its course in the calendar program (Line 9/A). Child B does not print it because it got stuck in a while loop (Line 16) and became an orphan.

C. How many times is the string “Aloha” (LINE B) getting printed? Briefly justify.

- Aloha does not get printed at all. Child A (created by the Parent) continues it course in the calendar program (Line 9/A). When Child A executes the execv() function it’s memory is overwritten with new code and data.

D. What do you expect will be the output generated by the snippet? Briefly explain your answer (mention any guarantees in the code for the order of execution between the active processes).

- I expect that Child B will print “Hello”; the Parent will print “Ciao” and then “Bye”. It is guaranteed that Child A would not print “Bye” because it continued it course in the calendar program (Line 9/A). It is also expected that Child B would not print “Bye” because it became an orphan after getting stuck in a while loop (Line 16).

E. Will there be any zombie or orphan processes created by this code? Briefly explain your answer.

- There will be an orphan that gets created by this code. Child B gets stuck in a while loop (Line 16). The parent will definitely finish before the Child. As a result, Child B becomes an orphan.