Operating Systems

Exercise 1: Write the following program and execute it

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
int i;
for(i = 0; i <3; i++)
{
        pid_t pid = fork();
        printf("I am process %d, My parent is %d\n",getpid(),getppid());
}
```

1. How many processes are generated by this code? Explain.

Exercise 2: Create a file called plan.txt and put the following two line in the file

This is plan number one
This is plan number two

Write the following program and execute it

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
int main()
{
        FILE *fp;
        char str[80];
        fp = fopen("plan.txt","r");
        if(fp == NULL)
        {
                fprintf(stderr,"Could not open file");
                exit(1);
        }
        fgets(str,80,fp);
        printf("Parent reads: %s\n",str);
        if(fork() == 0)
        {
                fgets(str,80,fp);
                printf("Child Reads: %s\n",str);
        }
        fclose(fp);
}
```

1. Explain why was the child process able to read from file that was opened in the parent process?
2. Change the code such that the file is opened for writing, and then the child process writes to the file the following line

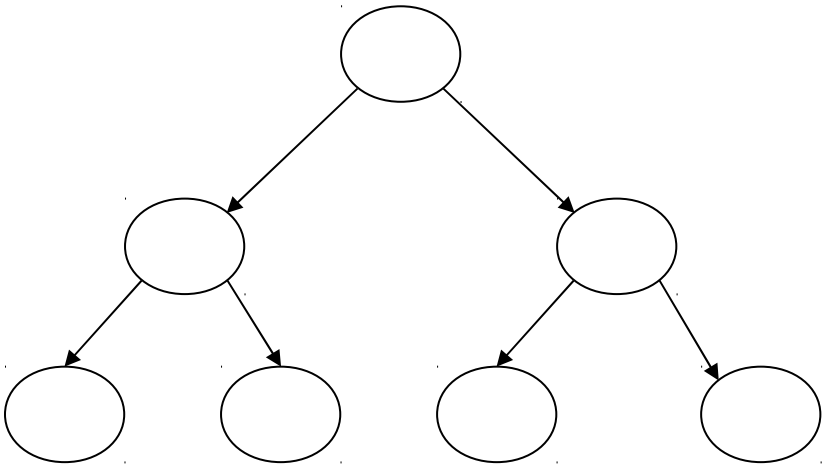
"This line written by child process"

While the parent process writes to the file the following line

"This line written by parent process"

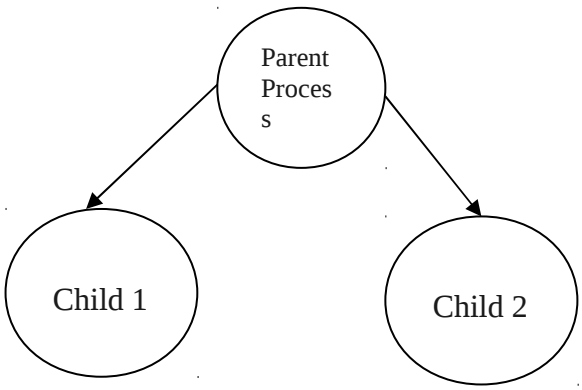
Run this program several times, what do you notice about the order in which these lines are printed? Do you see any problems with this?

Exercise 3: Write a C program to generate the following process tree. The root represents the parent process of your code.



Exercise 4: Write a C program to generate the following process tree. The root represents the parent process of your program. Child 1 and child 2 represent the first child and the second child of the parent process respectively. Each process should print its process ID and its parent process ID. The order of execution should be as follow:

a) Parent process → child 1 → child 2
b) Parent process → child 2 → child 1
c) Child 1 → parent process → child 2
d) Child 1 → child 2 → parent process



Exercise 5: Write a program using fork(), in which print name, roll no and birth-date of yours and your group mate separately in parent and child process created due to fork()

Exercise 6: In the exercise 5 question, use vfork() function to create two process and use exec() function in child process and do the same thing that is asked in exercise 5. Which is the proper place to use vfork instead of fork()?

Exercise 7: In the exercise 5 question, put a delay of 5 sec using sleep() function in child process and see the result you get.

Now use wait() function in parent process and see the result you get. Write down the result you get with or without wait() function.

Exercise 8: In the exercise 5, now put a delay of 5 sec in parent process and see the result. Explain and write the zombie process concept you found in this case.

Exercise 9: Write a collection of programs p1, p2, p3 such that they execute sequentially with the same process-id, and each program should also print its PID. The user should be able to invoke any combination of these programs, to achieve the required functionality.

For example consider three programs *twice*, *half*, *square* which accept only one integer as argument and does some specific operation.
$twice 10 prints **20** and some int which is its process-id as output
$square 10 prints **100** and some int which is its process-id as output
$half 10 prints **5** and some int which is its process-id as output

Now the user should be able to combine these programs in any combination to achieve the required result.

For example:
$twice square half twice half 10
should calculate *half(twice(half(square(twice(10)))))* and print **200** as result. It should also print the process ids of each program as it executes. **Note that the process-id printed by each of these programs should be the same, in this case.**
$square twice 2
should calculate *twice(square(2))* and print **8** as result, and the process id of square and twice, which should be the same.
The evaluation order is from left to right

Note that the last argument is the integer, and the remaining arguments are the programs to be invoked.

This should be generally applicable to any *n* number of processes, all of which are written by you. *minimum of three should be written*

pl p2 p3 ... pn arg_value