Exercise 1: Time slicing (or Time quantum)

Run the following program several times and observe the result of *time slice* used by UNIX.

```
#include <stdio.h>
#include <unistd.h>

#define MAX_ITR 100000

int main ()
{
    int i, j, pid;

    pid = fork ( );
    if ( pid == 0 ) {
        for ( i = 0; i < 20; i++ ) {
            for (j = 0; j < MAX_ITR; j++ );
            printf ("Child: %d\n", i) ;
        }
    }
    else {
        for ( i = 0; i < 20; i++ ) {
            for (j = 0; j < MAX_ITR; j++ );
            printf("Parent: %d\n", i);
        }
    }
}
```

1) What are your observations? Write down your observation about the time quantum for each process?
2) Modify the code in order to get **strict alternation** between the parent and child processes.
3) Modify the origin code in order to make the child process finishes before the parent process starts

Exercise 2: Files in child process

Create a file named "plan.txt" that contains the following two lines

```
This is plan number one
This is plan number two
```

Then, run the following code:

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

#define LINE_SIZ        80

int main()
{
        FILE *fp;
        char str[LINE_SIZ];

        fp = fopen("plan.txt", "r");
        if (fp == NULL) {
                fprintf(stderr, "File [plan.txt] not exist\n");
                exit(1);
        }

        /* read one line from file */
        fgets(str, LINE_SIZ, fp);
        printf("Parent reads: %s\n", str);

        if (fork() == 0)
        {
                /* read one line from file */
                fgets(str, LINE_SIZ, fp);
                printf("Child reads: %s\n", str);
        }

        fclose(fp);
}
```

1) Did you get a run-time error since the child process tries to read the file "plan.txt"?
2) What is your main conclusion?
3) Based on your conclusion, can you infer why each running program has **three** standard files: **stdin**, **stdout**, and **stderr.**

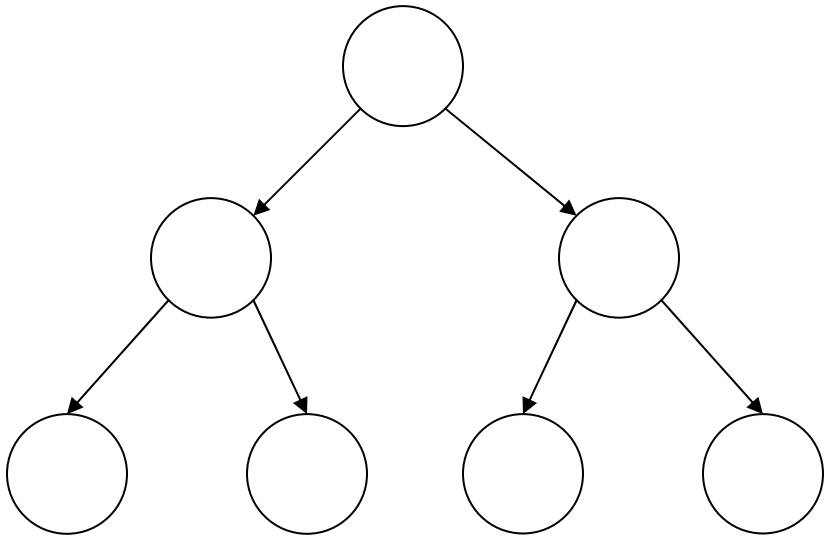
Exercise 3: Process Tree

Write a C program such that it forks a new process. Then the parent process and the child process should create one more process such that the program in all has four running processes. Each process should print its process ID and its parent process ID.

Draw process hierarchy starting from parent process.

Exercise 4: Process Tree

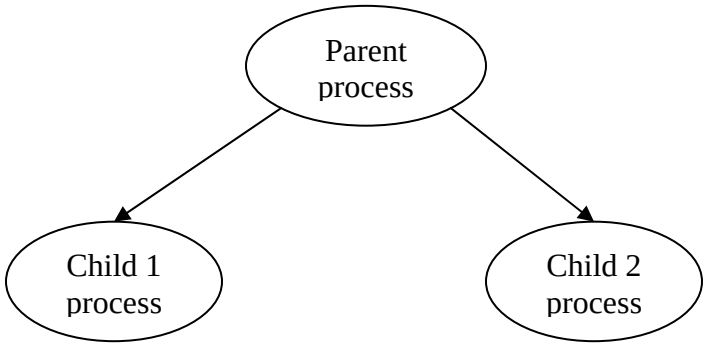
Write a C program to generate the following process tree. The root represents the parent process of your code.



Exercise 5: Process synchronization

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



The meaning of "order of execution" is that the first (in the order) process prints its process ID and its parent process ID before the other two processes. The second process in order prints its process ID and its parent process ID before the last one.