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

/*|Matheus Martins Batista (2019005687) - Sistemas Operacionais (COM120) |
  |Ciências da Computação (CCO) - EP01 - Exercício01 - 16/09/2021        |*/

int main(int argc, char *argv[]){

    int pid = 0, i;
    int pidpai, pidfilho;
    pidpai = getpid();
    pid = fork();
/*Conferir se o fork conseguiu criar um novo processo*/
    if(pid<0){
        perror("Fork");
        return 1;
    }
/*Verificar se o processo rodando é o pai ou filho com base no pid(fork do filho retorna 0)*/
/*Usar laço for para printar o status repetidamente do pai e do filho*/

    if (pid == 0){
        pidfilho = getpid();
        for(i=0;i<50;i++){
        printf("Sou processo filho!!! Meu PID é %d e o PID do meu pai é %d\n", pidfilho, pidpai);
        }
    }
    else{
        for(i=0;i<30;i++){
        printf("Sou o processo pai!!!Meu PID é %d\n", pidpai);
        }
    }
    return 0;
}
/*Rodando o código para loops de valores maiores e menores o comportamento se manteve:
o terminal imprime os PIDs de forma intercalada, iniciando com o pai e intercalando
com o filho. Aparentemente,ambos os processos estão printando ao mesmo tempo
(de forma simultânea) e a forma como o terminal expôs isso foi intercalando as impressões.
Essa percepção é devido ao momento em que o laço do processo pai termina, o processo filho
continua imprimindo o PID do filho e a recíproca também é válida.*/
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>

/*|Matheus Martins Batista (2019005687) - Sistemas Operacionais (COM120) |
  |Ciências da Computação (CCO) - EP01 - Exercício02 - 16/09/2021        |*/

int main(int argc, char *argv[]){

    int pid = 0, i;
    int pidpai, pidfilho, statusfilho;
    pidpai = getpid();
    pid = fork();
/*Conferir se o fork conseguiu criar um novo processo*/
    if(pid<0){
        perror("Fork");
        return 1;
    }
/*Verificar se o processo rodando é o pai ou filho com base no pid(fork do filho retorna 0)*/
/*Usar laço for para printar o status repetidamente do pai e do filho*/

    if (pid == 0){
        pidfilho = getpid();
        for(i=0;i<50;i++){
        printf("Sou processo filho!!! Meu PID é %d e o PID do meu pai é %d\n", pidfilho, pidpai);
        }
    }
/*incluir a função waitpid para forçar o pai a esperar o fiho terminar as impressões*/
    else{
        waitpid(pid, &statusfilho, 0);
        for(i=0;i<30;i++){
        printf("Sou o processo pai!!!Meu PID é %d\n", pidpai);
        }
    }
    return 0;
}
/*A variável pid recebe o PID do filho como retorno do fork(), ao rodar o pai pela primeira vez (fork do pai !=0) a condicional else executa a função waitpid, o processo pai espera a execução do filho(impressões do PID com laço for), após a execução do filho, o pai prossegue com as impressões e retorna o 0 (execução sem erros)*/
```

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

/*|Matheus Martins Batista (2019005687) - Sistemas Operacionais (COM120) |
  |Ciências da Computação (CCO) - EP01 - Exercício02 - 16/09/2021        |*/

int main(int argc, char **argv){

    int pid = fork();

    if(pid == 0){
        fork();
        fork();
    }
    else{
        fork();
    }

    printf("PID do processo %d\n", getpid());
    return 0;
}

/*Um printf foi utilizado para identificar o PID do processo em execução e, com isso,
foi possível verificar a presença de 6 processos criados ao decorrer da execução do
código. O primeiro filho (C1) é criado na linha 10, o pai prossegue com a execução
e entra na condicional else (linha 16), criando o filho 2 (C2). Por outro lado, o
C1 entra na condicional if (linha12) e cria mais dois processos (C3) e (C4). O
processo C3 é reponsável pela criação de C5. Logo, P->C1 e C2 | C1-> C3 e C4 | C3->C5.*/
```