# Práctica 2: Procesos

- Para resolver estos ejercicios se sugiere la utilización de alguno entorno POSIX compatible. Para esto, pueden utilizar alguna distribución de Linux o, en Windows, cygwin o msys2.
- Por otro lado, es recomendable buscar la referencias de las distintas funciones usando man, man 2.
- En general van a necesitar hacer los siguientes includes: sys/types.h, unistd.h, sys/wait.h

1. Considerar el siguiente código. ¿Cuál es el valor que se imprime de x? ¿Depende del orden de ejecución de los procesos?

```
int main(){
    int x = 0;
    int pid = fork();
    if(pid == 0){
        x++;
        exit(0);
    }
    else if(pid > 0){
        x++;
        wait(NULL);
        printf("%d",x);
        exit(0);
    }
}
```

2. Considerar el siguiente código y determinar el árbol de procesos que se genera.
*Sugerencia:* Pensar un caso más chico primero.

```
int main(){
    for(int i = 0; i< 4;i++)
        fork();
    return 0;
}
```

3. (a) Crear un programa en C de manera que:
   - El proceso padre, llamado Abraham, cree un proceso hijo llamado Homero
   - El proceso Homero cree dos hijos llamados Bart y Lisa (omitimos a Maggie por simplicidad)
   - Cada proceso debe imprimir por pantalla el nombre asociado al mismo.

   (b) Correr numerosas veces el programa anterior y determinar si el orden de la salida es siempre la misma. Si no lo es, ¿A qué puede deberse esto?

   (c) Si obtuviste distintos ordenes de outputs, modificá el programa realizado para que el orden obtenido sea el mismo que el de creación

   (d) ¿En cuál de las dos implementaciones es posible que se generen zombies?

4. Determinar si *"Proceso1-Terminaron todos-Proceso0-Proceso4-Proceso2-Proceso3"* es una secuencias de outputs posible en el siguiente programa. Si lo es, modificá el código para que efectivamente, el mensaje "Terminaron todos"se imprima cuando realmente eso ocurra.

```
int maint(){
    int pid, status;
    for(int i = 0;i<5;i++){
        pid = fork();
        if(pid==0){
            cout << "Proceso" << i << endl;
            exit(0);
        }
    }
    if(pid > 0){
       wait(&status);
       cout << "Terminaron todos " << endl;
       exit(0);


}
```

5. Crear un programa que permita al usuario ejecutar comandos básicos de Linux como *mkdir, ls, pwd, etc.* Tener en cuenta que estos podrían contener parámetros, por ejemplo *ls-l. Sugerencia: Considerar la función execvp()*

6. (a) Crear un programa que cree n procesos de forma que cada uno tenga exactamente un hijo (excepto el último)

   (b) Modificar el esquema anterior, de manera que los nodos pares se "pinten"de color negro y los impares de rojo. Para esto, considerar que el número de cada nodo se corresponde con el del orden de creación y el ancestro común de todos recibe el 0.
   Finalmente, cada nodo debe informar su color por pantalla.

7. Crear un programa tal que en ejecución genere un árbol de procesos que sea binario y completo de altura n.

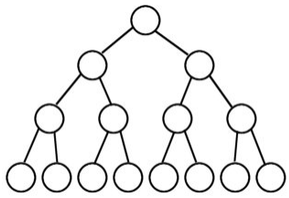

Figura 1: Ejemplo para n = 3