PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ
FACULTAD DE CIENCIAS E INGENIERÍA

SISTEMAS OPERATIVOS
Semestre 2021-1
Laboratorio 1

1) (12 puntos) En el material para este laboratorio se encuentra el siguiente código, que construye un árbol binario.

```c
/* ************************************************************/
/* binarytree.c (c) 2010 Alejandro T. Bello Ruiz, GPL-like     */
/* *********** ************************************************/
#include <stdio.h>
#include <unistd.h>
#include <math.h>
#include <stdlib.h>
#include <string.h>
#include <sys/wait.h>

double final;

void crea_arbol(int);

int main(int narg, char *argv[])
{  int n;

   n=atoi(argv[1]);
   final=pow(2,(n-1));
   crea_arbol(1);
   return 0;
}

void crea_arbol(int x)
{  char cadena[60];

   sprintf(cadena,"Soy el proceso %d con pid %d y ppid %d\n",x,getpid(),getppid());
   write(1,cadena,strlen(cadena));
   if(x >= final) return;
   if(!fork()) { crea_arbol(2*x); exit(0); }
   if(!fork()) { crea_arbol(2*x+1);exit(0);}
   wait(NULL);
   wait(NULL);
}
```

Se propone como alternativa el siguiente programa:

```c
/* ************************************************************/
/* binarytree.c (c) 2021 Alejandro T. Bello Ruiz, GPL-like      */
/* *********** ************************************************ */
#include <stdio.h>
#include <unistd.h>
#include <math.h>
#include <stdlib.h>
#include <string.h>
#include <sys/wait.h>

double final;

void crea_arbol(int);

int main(int narg, char *argv[])
{
   final=atoi(argv[1]);
   crea_arbol(0);
   return 0;
}

void crea_arbol(int nivel)
{  int k;

   nivel++;
   for(k=0;k<2;k++)
      if(!fork()) {
         if(nivel < final) crea_arbol(nivel);
         else pause();
      }
   wait(NULL);
   wait(NULL);

}
```

Todos los procesos del programa de arriba se llegan a bloquear, de forma que si deseamos ver el árbol de procesos, lo mejor es lanzarlo en segundo plano (*background*) desde una terminal, tal como se muestra a continuación:

```
alejandro@abdebian:2021-1$ ./binarytree1 3&
[1] 14559
alejandro@abdebian:2021-1$ pstree -p 14559
binarytree1(14559)─┬─binarytree1(14562)─┬─binarytree1(14564)─┬─binarytree1(14569)
                   │                    │                    └─binarytree1(14573)
                   │                    └─binarytree1(14568)─┬─binarytree1(14575)
                   │                                         └─binarytree1(14576)
                   └─binarytree1(14563)─┬─binarytree1(14565)─┬─binarytree1(14570)
                                        │                    └─binarytree1(14572)
                                        └─binarytree1(14566)─┬─binarytree1(14571)
                                                             └─binarytree1(14574)
alejandro@abdebian:2021-1$ killall binarytree1
[1]+  Terminado               ./binarytree1 3
alejandro@abdebian:2021-1$
```

Se le solicita lo siguiente:

a) (5 puntos) (***binarytree2.c***) Modificar el programa *binarytree1.c* (proporcionado en el archivo fuentes.zip) para que la orden pstree sea lanzado desde el mismo programa. El que debe lanzarlo debe ser el último proceso creado, para asegurarse que en la salida se encuentran todos los procesos. No debe usar *pipes*. No debe usar la función de librería *sleep*, pues los procesos ya se encuentran bloqueados. La salida debe ser análoga ala siguiente:

```
alejandro@abdebian:2021-1$ ./binarytree2 3
binarytree2(16091)─┬─binarytree2(16092)─┬─binarytree2(16094)─┬─binarytree2(16098)
                   │                    │                    └─binarytree2(16102)
                   │                    └─binarytree2(16096)─┬─binarytree2(16099)
                   │                                         └─binarytree2(16103)
                   └─binarytree2(16093)─┬─binarytree2(16095)─┬─binarytree2(16100)
                                        │                    └─binarytree2(16104)
                                        └─binarytree2(16097)─┬─binarytree2(16101)
                                                             └─binarytree2(16105)───sh(16106)───pstree(16107)
Terminado
```

Observe que es el último proceso creado quien ejecuta la orden pstree.

b) (7 puntos) (***binarytree3.c)*** Modifique el programa *binarytree2.c* para que el proceso padre vaya eliminando (*kill*) los procesos del árbol, por niveles, hasta quedar él solo y terminar. Cada vez que elimina los procesos de un nivel, debe ejecutar la orden *pstree*. Para este ejercicio basta con emplear un solo *pipe*. La salida debe se análoga a la siguiente:

```
alejandro@abdebian:2021-1$ ./binarytree3 3
binarytree3(15966)─┬─binarytree3(15967)─┬─binarytree3(15972)─┬─binarytree3(15976)
                   │                    │                    └─binarytree3(15979)
                   │                    └─binarytree3(15975)─┬─binarytree3(15980)
                   │                                         └─binarytree3(15981)
                   ├─binarytree3(15968)─┬─binarytree3(15970)─┬─binarytree3(15973)
                   │                    │                    └─binarytree3(15977)
                   │                    └─binarytree3(15971)─┬─binarytree3(15974)
                   │                                         └─binarytree3(15978)
                   └─sh(15969)───pstree(15982)
Eliminando procesos del nivel 3
binarytree3(15966)─┬─binarytree3(15967)─┬─binarytree3(15972)
                   │                    └─binarytree3(15975)
                   ├─binarytree3(15968)─┬─binarytree3(15970)
                   │                    └─binarytree3(15971)
                   └─sh(15983)───pstree(15984)
Eliminando procesos del nivel 2
binarytree3(15966)─┬─binarytree3(15967)
                   ├─binarytree3(15968)
                   └─sh(15985)───pstree(15986)
Eliminando procesos del nivel 1
binarytree3(15966)───sh(15987)───pstree(15988)
```

2) (8 puntos) (***anilloFib.c***) This exercise develops a simple application in which processes generate a sequence of Fibonacci numbers on the ring. The next number in a Fibonacci sequence is the sum of the previous two numbers in the sequence.

In this question, the processes pass information in character string format. The original parent outputs the string "1 1" representing the first two Fibonacci numbers to standard output, sending the string to the next process. The other processes read a string from standard input, decode the string, calculate the next Fibonacci number, and write to standard output a string representing the previous Fibonacci number and the one just calculated. Each process then writes the result of its calculation to standard error and exits. The original parent exits after receiving a string and displaying the numbers received.

Start with the origina *anillo.c* program and replace the fprintf with code to read two integers from standard input in the string format described below, calculate the next integer in a Fibonacci sequence, and write the result to standard output.

1. Each string is the ASCII representation of two integers separated by a single blank.

2. The original parent writes out the string "1 1", representing two ones and then reads a string. Be sure to send the string terminator.

3. All other processes first read a string and then write a string.

4. Fibonacci numbers satisfy the formula fib(n+1) = fib(n) + fib(n-1). Each process receives two numbers (e.g., a followed by b), calculates c = a + b and writes b followed by c as a null-terminated string. (The b and c values should be written as strings separated by a single blank.)

5. After sending the Fibonacci number to standard error, the process exits. Try to write the program in such a way that it handles the largest possible number of processes and still calculates the Fibonacci numbers correctly.

Notes: The program should be able to calculate Fib(46)=1,836,311,903, using 45 processes or Fib(47)=2,971,215,073, using 46 processes. It may even be able to calculate Fib(78)=8,944,394,323,791,464, using 77 processes.

```
alejandro@abdebian:2021-1$ ./anilloFib 78
El fibonacci de 78 es 8944394323791464
alejandro@abdebian:2021-1$
```

Porf. Alejandro T. Bello Ruiz