UNIVERSIDAD SAN CARLOS DE GUATEMALA

CENTRO UNIVERSITARIO DE OCCIDENTE

DIVISIÓN CIENCIAS DE LA INGENIERÍA

CARRERA DE INGENIERÍA CIENCIAS Y SISTEMAS

LABORATORIO DE SISTEMAS OPERATIVOS 1

ING.: FRANCISCO ROJAS

ESTUDIANTE: Sergio Daniel Cifuentes Argueta - 201730705

Árbol de Procesos

FECHA: Marzo de 2021

INTRODUCCIÓN

Linux es un sistema operativo muy popular en el área de la informática, además de ser multitarea y multiusuario, esto implica que múltiples procesos pueden operar simultáneamente sin interferirse unos con los otros. Esto nos lleva a la necesidad de conocer de estos procesos y cómo administrarlos en linux. En esta práctica se estará creando una estructura de varios procesos con jerarquía de padres e hijos. De esta manera se aprenderá de la simplicidad del sistema multitarea del sistema operativo.

OBJETIVOS

General:

- Representar con una estructura de árbol, varios procesos ejecutándose en el sistema operativo según la cantidad deseada.

Específicos:

- Cambiar el color de una parte gráfica en un proceso.
- Obtener inputs del usuario y así modificar los procesos.
- Imprimir en un archivo de texto el estado actual de un árbol.

DESCRIPCIÓN DE LA PRÁCTICA

Esta práctica consiste en la creación de un árbol de procesos según el input que ingrese el usuario. Cada uno de estos procesos serán la representación de un árbol de verdad(hoja, rama o tronco) y tendra la tarea de contolar el cambio de color de la representacion grafica. El usuario tendrá la capacidad de realizar tres diferentes comandos.

El primero es el "P" que representa la creación o modificación de una planta. El comando estará construido de la siguiente forma: *(P,1,2,3)*. Donde el primer parámetro es obligatorio e indica el id de la planta. Si este id ya existe entonces solo se editará la planta ya existente. El segundo parámetro es opcional y representa la cantidad de ramas que tendrá el trono con el id indicado. Si no viene un segundo parámetro, no tendrá ramas la planta. Si la planta ya existe y el número de ramas es mayor a los indicados en el comando entonces le eliminarán las ramas necesarias para cumplir con lo querido. Por último el tercer parámetro indica cuantas hojas tendrán las ramas y si este no se indica entonces las ramas no tendrán hojas. Si la planta ya existe y el número de hojas es mayor a los indicados en el comando entonces le eliminarán las hojas de las ramas necesarias para cumplir con lo querido.

El segundo comando es el de "M" (mostrar) que indica que el usuario desea ver el árbol indicado. Este comando estará construido de la siguiente manera *(M,1)*. El primer parámetro indica el id del árbol que se desea mostrar.

Por último el “I” imprime la planta en archivo de texto con número de procesos y el árbol de procesos. Este comando estará construido de la siguiente manera *(I,1).* El primer parámetro indica el id del árbol que se desea imprimir.

Para el cambio de colores se manejara de la sigueinte manera:

Trono	Negro y Gris
Rama	Verde y Cafe
Hoja	Múltiples colores

Para el input del usuario existirán dos formas; La interactiva que consiste en ingresar el comando mediante la interfaz de la aplicación, y el modo texto en donde se carga un archivo con los comandos.

Hardware

- Laptop Hp Core i5 7th Gen.

Software

- **Ubuntu 20.04:** es una distribución GNU/Linux que ofrece un interesante sistema operativo para equipos de escritorio y servidores en el ámbito educativo. Es una distribución basada en Debian cuyas principales características son: Facilidad de manejo. Actualizaciones frecuentes.
- **C++ : e**s un lenguaje de programación diseñado en 1979 por Bjarne Stroustrup. La intención de su creación fue extender al lenguaje de programación C mecanismos que

permiten la manipulación de objetos. En ese sentido, desde el punto de vista de los lenguajes orientados a objetos, C++ es un lenguaje híbrido.

- **Fork()** Es una herramienta para Windows que nos permitirá crear dispositivos de arranque de diversa índole a partir de unidades de disco externas como pendrives USB o tarjetas SD. Su versatilidad permite tanto formatear una unidad como instalar imágenes de disco de Linux, Windows e incluso FreeDOS, integrado en la herramienta.
- **Qt Creator:** es un entorno de desarrollo integrado multiplataforma que puede usar para crear aplicaciones o modificar las ya existentes. Qt Creator está incluido en la instalación de escritorio de ArcGIS AppStudio para ayudarle a crear aplicaciones.
- **pstree :** es un comando poderoso y útil para mostrar procesos en ejecución en Linux. Al igual que su ps compañero, muestra todos los procesos que se están ejecutando actualmente en su sistema conectado. ... Esto permite la eliminación precisa de procesos problemáticos o fuera de control con el comando kill.
- **ps aux:** muestra todos los procesos pertenecientes al usuario llamado "x", ya que no suele existir un usuario llamado "x" ps lo interpreta como el comando "ps aux" e imprime una advertencia :D. Por defecto ps sólo muestra los procesos con el mismo user id efectivo (EUID) que el del usuario que lo ejecuta.
- **pkill :** es una utilidad de la línea de comandos escrita originalmente para ser usada con el sistema operativo Solaris. Desde entonces, se ha implementado para Linux y BSD. Como los comandos kill y killall, pkill se usa para enviar señales. El comando pkill permite usar expresiones regulares y otros criterios de selección.

Pasos a realizar

- Crear interfaz de la aplicación
- Leer input de los usuarios
- Crear procesos según número de ramas y hojas
- Asignarle la tarea de cambiar colores a cada proceso
- Mostrar el cambio de colores de manera gráfica
- Imprimir el estado del proceso en un archivo

MARCO TEÓRICO

Procesos de linux

Los procesos en Linux, o en cualquier sistema operativo *nix, son creados en base a un proceso ya existente mediante un mecanismo de clonación, o «*fork*«. Hoy hablaremos específicamente de gestión de procesos en linux.

Un proceso en Linux genera un nuevo proceso para que realice una tarea determinada, y este nuevo proceso es considerado proceso «*hijo*» del proceso anterior, al que llamaremos «*padre*«.

Esta estructura de procesos padres e hijos forman un árbol jerárquico de procesos, en los que podemos distinguir a hilos del kernel, al proceso init, y al resto de los procesos del sistema, descolgados de algún otro proceso, lo que nos da una idea de qué proceso generó a cuál otro.

PID

El PID es un número entero que identifica unívocamente a cada proceso en una tabla de procesos administrada por el kernel Linux. Esta tabla de procesos mantiene una entrada por cada uno de los procesos que están en ejecución en el sistema en el momento actual. Esa tabla es, precisamente, la que se consulta con comandos como ps o pstree.

Uno de los datos almacenados, es el pid, o process-id, utilizado, como hemos visto en un artículo anterior, para facilitarnos la terminación de procesos que no responden, o para que el sistema operativo pueda comunicar procesos usando IPC, entre otras tareas.

Kernel Multitarea

El kernel Linux es un kernel de sistema operativo que permite la multitarea. Hay que tener en cuenta que el procesador solamente puede estar procesando un proceso a la vez en cada uno de sus núcleos, o «cores».

¿Cómo logra Linux simular la multitarea? ¿Cómo es posible, de esta forma, que podamos estar ejecutando varias aplicaciones «simultáneamente» con un solo core?

La respuesta es simple. El kernel del sistema operativo le da un tiempo de procesamiento a un proceso, luego lo retira de los registros del procesador, e introduce a otro proceso en su lugar. Este intercambio se lleva a cabo con una frecuencia altísima que permite simular una multitarea real a nivel de usuario.

Multiproceso, Multitarea y Multiusuario

Multiproceso, multitarea y multiusuario son características de los sistemas operativos. A continuación se pasará a estudiar estos conceptos en profundidad:

Multiproceso. Un sistema operativo multiproceso es aquel que puede ejecutar varios procesos de forma concurrente (a la vez). Para que se puedan ejecutar varios procesos a la vez es necesario tener un sistema multiprocesador (con varios procesadores).

El objetivo de utilizar un sistema con varios procesadores no es ni más ni menos que aumentar la potencia de cálculo y por lo tanto rendimiento del mismo. El sistema va repartiendo la carga de trabajo entre los procesadores existentes y también tendrá que gestionar la memoria para poder repartirla entre dichos procesadores. Generalmente

Los sistemas multiprocesador se utilizan en workstations, servidores, etc.

El realizar un sistema operativa que pueda trabajar con varios procesadores de forma concurrente es una tarea complicada y generalmente se diseña para que trabajen de varias formas:

- **Forma asimétrica.** Se designa un procesador como el procesador master (maestro) y los demás serán slave (esclavos) de este. En el procesador maestra se ejecuta el sistema operativo y se encargará de repartir el trabajo entre los demás procesadores esclavos. Este sistema tiene la ventaja de que es más simple de implementar ya

  que se centraliza la gestión en un procesador. Por el contrario, el que existan procesadores que realicen distinto trabajo hace que el sistema no sea tan eficiente como un sistema operativo que trabaje de forma simétrica.
- **Forma simétrica.** En este tipo de sistema operativo todos los procesadores realizan las mismas funciones. Es más difícil de implementar pero son más eficientes que los sistemas operativos multiprocesos asimétricas. El poder balancear la carga de trabajo entre todos los procesadores existentes hace que el tiempo de inactividad de los mismos sea mucho menor y por lo tanto la

productividad mucho más alta. Otra ventaja es que si un procesador falla. Gracias al balanceo de carga, los demás procesadores se hacen cargo de las tareas del procesador que ha fallado.

- **Multitarea** En un sistema operativo multitarea, varios procesos se pueden estar ejecutando aparentemente al mismo tiempo sin que el usuario la perciba. La gran mayoría de sistemas operativos actuales son multitarea. Un sistema multitarea permite a la vez estar escuchando música, navegando por internet y realizando una videoconferencia. El sistema operativo fracciona el tiempo de CPU y lo va repartiendo entre los procesos que lo necesitan de la mejor forma posible. Además de la multitarea aparece el concepto de multihilo.
- **Multihilo.** En ocasiones, un proceso puede tener varios hilos de ejecución de tal manera que un proceso pueda ejecutar varias tareas a la vez El multihilo se utiliza a veces por eficiencia, debido a que el creer muchos procesos implica la asignación de muchos recursos mientras que muchos hilos pueden compartir los recursos y memoria de un proceso. El multihilo evita además la pérdida de tiempo en cambios de contexto al ejecutarse todos los hilos en el mismo contexto.
- **Multiusuario**. Un sistema operativo multiusuario es un sistema que puede dar servicio a varios usuarios de forma simultánea. Hay que tener en cuenta que un sistema con varias cuentas de usuario no es necesariamente un sistema multiusuario. Por ejemplo, las versiones domésticas de los sistemas Windows permiten tener varias cuentas de usuario en el mismo sistema operativo pero no permiten que haya varios usuarios trabajando en el sistema al mismo tiempo Las versiones Server de Windows si permiten esta característica a

través de terminal server. Por el contrario, las versiones Linux si son multiusuarios desde hace mucho tiempo.

Llamada al sistema fork

Los procesos en Linux tienen una estructura jerárquica, es decir, un procesopadre puede crear un nuevo proceso hijo y así sucesivamente. La forma enque un proceso arranca a otro es mediante una llamada al sistema fork oclone. Cuando se hace un fork, se crea un nuevo task_struct a partir deltask_struct del proceso padre. Al hijo se le asigna un PID propio y se lecopian las variables del proceso padre. Sin embargo, vemos como en lallamada al sistema clone (figura 2) el task_struct del proceso padre se copiay se deja tal cual, por lo que el hijo tendrá el mismo PID que el procesopadre y obtendrá (físicamente) las mismas variables que el proceso padre.El proceso hijo creado es una copia del padre (mismas instrucciones, mismamemoria). Lo normal es que a continuación el hijo ejecute una llamada alsistema exec. En cuanto al valor devuelto por el fork, se trata de un valornumérico que depende tanto de si el fork se ha ejecutado correctamentecomo de si nos encontramos en el proceso padre o en el hijo.

¿Cómo se ejecuta la llamada al sistema fork?

¿Qué secuencia de eventostiene lugar desde que se encuentra una llamada al sistema fork en el códigohasta que se empieza a ejecutar do_fork, la función principal del archivofork.c? Veámoslo paso a paso:

1. La función fork de la librería libc coloca los parámetros de la llamadaen los registros del procesador y se ejecuta la instrucción INT 0x80.
2. Se conmuta a modo núcleo y, mediante las tablas IDT y GDT, se llamaa la función sys_call.

3. La función sys_call busca en la sys_call_table la dirección de lallamada al sistema sys_fork.
4. La función sys_fork llama a la función do_fork, que es la funciónprincipal del archivo fork.

DESARROLLO DEL PROYECTO

Crear interfaz de la aplicación

Para Crear la interfaz de la aplicación en c++ se decidio trabajar en el software Qt Creator, que contiene muchos asi que a continuacion indicare el proceso de instalacio.

Para empezar vamos a instalar Build Essential, si es que no lo tienes ya instalado. Este es un paquete que va a permitirnos a los usuarios instalar y utilizar herramientas de c ++ en Ubuntu. Para proceder a la instalación, abrimos una terminal (Ctrl+Alt+T) y primero actualizaremos el software disponible para después instalar el paquete escribiendo:

sudo apt update; sudo apt install build-essential

Si no tienes instalado el paquete Qt Creator que contiene la IU y las herramientas de línea de comandos para la creación y ejecución del proyecto Qt, escribe en la misma terminal:

sudo apt install qtcreator

Si quieres que Qt5 se use como la versión predeterminada de Qt Creator, ejecuta el siguiente comando:

sudo apt install qt5-default

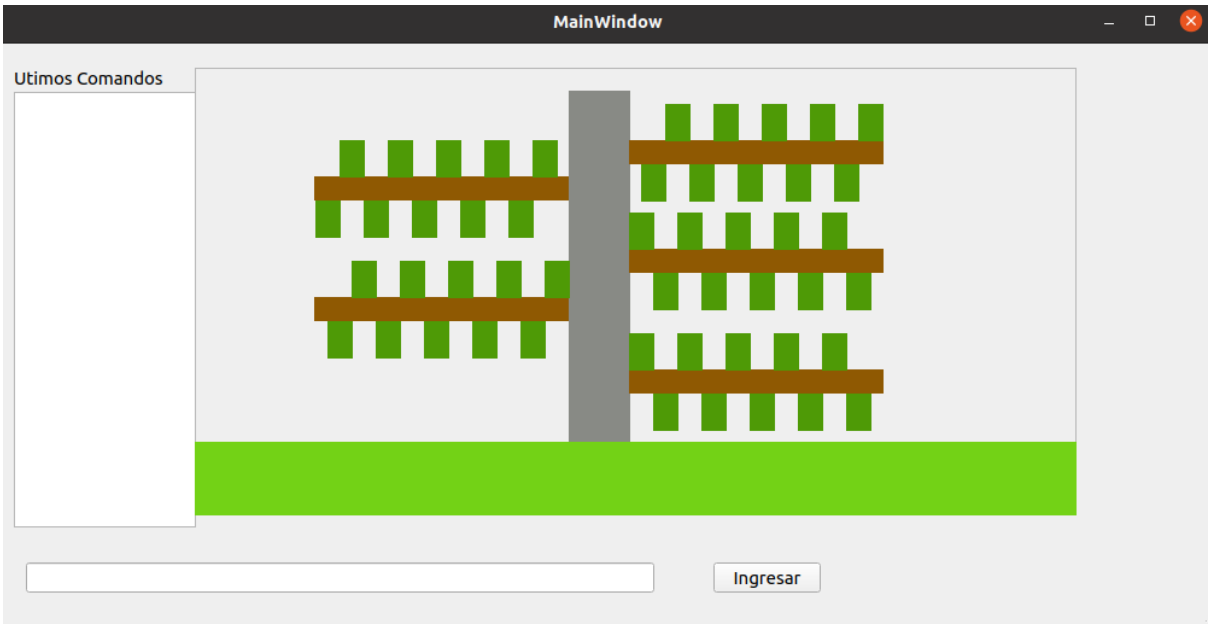
Para implementar proyectos más complejos, habrá que instalar la documentación y los ejemplos de Qt. Esto se puede hacer escribiendo en la terminal:

sudo apt-get install qt5-doc qtbase5-examples qtbase5-doc-html

El paquete *qt5-doc* contiene la documentación API de Qt 5. *Qtbase5-examples* contiene los ejemplos de Qt Base 5 y *qtbase5-doc-html* contiene la documentación HTML para las bibliotecas base de Qt 5.

Interfaz

Como se planea mostrar de manera gráfica el árbol se planteó esta interfaz.



En donde en la parte inferior, el usuario tendrá un espacio para ingresar el comando deseado. En la parte de la izquierda se guardará una historia de los comandos que se utilizaron y en la parte central se ve el árbol con las 5 ramas y 10 hojas.

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>MainWindow</class>
 <widget class="QMainWindow" name="MainWindow">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>1004</width>
    <height>487</height>
   </rect>
  </property>
  <property name="windowTitle">
   <string>MainWindow</string>
  </property>
  <widget class="QWidget" name="centralwidget">
   <widget class="QPushButton" name="pushButton">
    <property name="geometry">
     <rect>
      <x>590</x>
      <y>430</y>
      <width>89</width>
      <height>25</height>
     </rect>
    </property>
    <property name="text">
     <string>Ingresar</string>
    </property>
   </widget>
   <widget class="QLineEdit" name="input">
    <property name="geometry">
     <rect>
      <x>20</x>
```

```
</widget>
<widget class="QLabel" name="label">
 <property name="geometry">
  <rect>
   <x>10</x>
   <y>20</y>
   <width>131</width>
   <height>17</height>
  </rect>
 </property>
 <property name="text">
  <string>Utimos Comandos</string>
 </property>
</widget>
<widget class="QFrame" name="frame">
 <property name="geometry">
  <rect>
   <x>160</x>
   <y>20</y>
   <width>731</width>
   <height>371</height>
  </rect>
 </property>
 <property name="autoFillBackground">
  <bool>true</bool>
 </property>
 <property name="frameShape">
  <enum>QFrame::StyledPanel</enum>
 </property>
 <property name="frameShadow">
  <enum>QFrame::Raised</enum>
 </property>
 <widget class="QMdiArea" name="tierra">
  <property name="geometry">
```

Código de la interfaz Qt

Leer input de los usuarios

Con el textarea ya mencionado y además el boton para ingresar el comando, el texto ingresado por el usuario será analizado por el backend. Primero se verificara que clase de comando es. Por ejemplo si el comando es P se hace un split del texto para obtener el numero de hojas y ramas

```
if(result.at(0).substr(1,1)=="P"){
    int hojas=0;
    int ramas=0;
    int id=0;

    if(result.size()>=3){
        stringstream geek(result.at(1));
         geek >> id;
        if(result.size()==4){
            stringstream geek(result.at(2));
            geek >> ramas;

            vector<string> result2;
            stringstream s_stream(result.at(3)); //create string stream from the string
            while(s_stream.good()) {
               string substr;
               getline(s_stream, substr, ')'); //get first string delimited by comma
               result2.push_back(substr);
            }
            stringstream geek2(result2.at(0));
            geek2 >> hojas;
        }else{
            vector<string> result2;
            stringstream s_stream(result.at(2)); //create string stream from the string
            while(s_stream.good()) {
               string substr;
               getline(s_stream, substr, ')'); //get first string delimited by comma
               result2.push_back(substr);
            }
            stringstream geek(result2.at(0));
            geek >> ramas;
        }
```

```
}else if(result.at(0).substr(1,1)=="I"){
    int id;
    vector<string> result2;
    stringstream s_stream(result.at(1)); //create string stream from the string
    while(s_stream.good()) {
       string substr;
       getline(s_stream, substr, ')'); //get first string delimited by comma
       result2.push_back(substr);
    }
    stringstream geek(result2.at(0));
    geek >> id;
    imprimirEstado(id);



}else{
```

Si el comando empieza con una m, simplemente se cambiará el valor de planta de un componente que veremos más adelante.

```
}else if(result.at(0).substr(1,1)=="M"){
    int id;
    vector<string> result2;
    stringstream s_stream(result.at(1)); //create string stream from the string
    while(s_stream.good()) {
       string substr;
       getline(s_stream, substr, ')'); //get first string delimited by comma
       result2.push_back(substr);
    }
    stringstream geek(result2.at(0));
    geek >> id;
    threadLeer->planta=id;
```

Si el comando empieza con I se llamará una función de imprimir con el id del que se indicó.

```
    int id;
    vector<string> result2;
    stringstream s_stream(result.at(1)); //create string stream from the string
    while(s_stream.good()) {
       string substr;
       getline(s_stream, substr, ')'); //get first string delimited by comma
       result2.push_back(substr);
    }
    stringstream geek(result2.at(0));
    geek >> id;
    imprimirEstado(id);
```

```
ofstream file;
std::ostringstream s;
s<< "ImprimirPlanta"<<id;

char input[50];
        sprintf(input,"pstree |grep Hoja%d",id);


std::array<char, 128> buffer;
    std::string result;
std::unique_ptr<FILE, decltype(&pclose)> pipe(popen(input, "r"), pclose);
    if (!pipe) {
        throw std::runtime_error("popen() failed!");
    }
    while (fgets(buffer.data(), buffer.size(), pipe.get()) != nullptr) {
        result += buffer.data();
    }




file.open(s.str());
file <<result ;
file.close();
```

Crear procesos según número de ramas y hojas

Para este objetivo utilizaremos objetos para facilitar y que sea de mas facil comprension. Contamos con objetos planta, hoja y rama. El numero de planta obtendra el id, el numero de ramas y el numero de hojas como se muestra en la figura.

```
planta(int plantaId,int numeroRamas, int numeroHojas) {
  id=plantaId;
  p=fork();

  if(p==0){


    std::ostringstream s;
     s<< "Planta"<<id;
        std:: string processName(s.str());
        int n = processName.length();
        char processNameChar[n + 1];
        strcpy(processNameChar, processName.c_str());
        crearCarpteta(processName);
      prctl(PR_SET_NAME,processNameChar,NULL,NULL,NULL);
      for (int i=0;i<numeroRamas ;i++ ) {
          if(i>0&&ramas[i-1].getP()==0){
              break;
          }
          qDebug("\nCreando Rama%d\n",i);

          ramas[i]=rama(i+1,numeroHojas,id,processName);
      }
      cambiarColor(processName);

      sleep(10);
  }
}
```

Luego de esto, el constructor llama a la función que se encarga de crear un directorio con el nombre de la planta, esto para poder llevar a cabo el cambio de valores.

```
void crearCarpteta(string nombrePlanta){
    std::ostringstream s2;
    s2<< nombrePlanta;
    std:: string carpeta(s2.str());
    int n2 = carpeta.length();
    char carpetaChar[n2 + 1];
    strcpy(carpetaChar, carpeta.c_str());

    mkdir(carpetaChar,0777);
}
```

Luego de crear la carpeta, según el número indicado de ramas, se instalan objetos ramas. Estos objetos tendrán como parametro el id y el número de ramas que tendrán, como se muestra en la siguiente imagen.

```
rama(int idRama,int numeroHojas,int idPlanta,std::string nombrePlanta) {    // Constructor
  id=idRama;
  p=fork();
  if(p==0){
    std::ostringstream s;
     s<< "Rama"<<idPlanta<<"_"<<id;
        std:: string processName(s.str());
        int n = processName.length();
        char processNameChar[n + 1];
        strcpy(processNameChar, processName.c_str());
        crearCarpteta(nombrePlanta,processName);
      prctl(PR_SET_NAME,processNameChar,NULL,NULL,NULL);
      for (int i=0;i<numeroHojas ;i++ ) {
          if(i>0&&hojas[i-1].getP()==0){
              break;
          }
          hojas[i]=hoja(i+1,idPlanta,id,nombrePlanta,processName);
      }
      cambiarColor(nombrePlanta,processName);

      sleep(10);
  }

}
```

Como puede ver el constructor de la rama y de la planta son muy similares esto es porque hacen lo mismo ya que deben de crear procesos hijos según lo indicado. Finalmente el constructor de la hoja se verá de la siguiente manera

```
hoja(int idhoja,int idPlanta, int idRama,std::string nombrePlanta,std::string nombreRama) {
  id=idhoja;
  p=fork();
  if(p==0){
    std::ostringstream s;
     s<< "Hoja"<<idPlanta<<"_"<<idRama<<"_"<<id;
        std:: string processName(s.str());
        int n = processName.length();
        char processNameChar[n + 1];
        strcpy(processNameChar, processName.c_str());
        crearCarpteta(nombrePlanta,nombreRama,processName);
      prctl(PR_SET_NAME,processNameChar,NULL,NULL,NULL);
      cambiarColor(nombrePlanta,nombreRama, processName);

      sleep(10);
  }

}
```

Asignarle la tarea de cambiar colores a cada proceso

Despues de que cada proceso cree los hijos indicados, tendra la tarea de cambiar de color a su representacion grafica, esto ocurrira hasta que se elimine el proceso. Como puede ver en los constructores anteriores se aprecia una función llamada cambiar color. Esta función sera un ciclo

while y se encargará de cambiar el número de un archivo según lo deseado. Esta función se ve de la siguiente manera.

```
void cambiarColor(string processName){
    int colorActual=INT_NEGRO;
    while(true){

        ofstream file;
        file.open(processName+"/"+"color");
        file << colorActual;
        file.close();
        colorActual=getRgbTronco(colorActual);
        sleep(5);
    }
}
planta() {      // Constructor
```

En donde se abre un ofstream del archivo indicado y se ingresa el color deseado, es importante resaltar la función de getRgb() ya que este se encarga de la administración de los colores.

Manejador de colores

Este archivo contendrá funciones y variables estáticas con el objetivo de facilitar los colores. Además hace mucho más fácil la actualización de la paleta de colores si en algún momento se necesita.

```
static string convertint(int lastRgb){

    if(lastRgb==INT_VERDE)return VERDE;
    if(lastRgb==INT_NEGRO)return NEGRO;
    if(lastRgb==INT_GRIS)return GRIS;
    if(lastRgb==INT_CAFE)return CAFE;
    if(lastRgb==INT_NARANJA)return NARANJA;
    if(lastRgb==INT_ROJO)return ROJO;
    if(lastRgb==INT_AMARILLO)return AMARILLO;

    return NEGRO;
}
```

```
static int getRgbHoja(int lastRgb){

    if(lastRgb==INT_VERDE)return INT_NARANJA;
    if(lastRgb==INT_NARANJA)return INT_ROJO;
    if(lastRgb==INT_ROJO)return INT_AMARILLO;
    return INT_VERDE;
}
```

```
static const string NEGRO="Black";
static const string GRIS="Gray";
static const string VERDE="Green";
static const string CAFE="Brown";
static const string NARANJA="Orange";
static const string ROJO="Red";
static const string AMARILLO="Yellow";
static const int INT_NEGRO=1;
static const int INT_GRIS=2;
static const int INT_VERDE=3;
static const int INT_CAFE=4;
static const int INT_NARANJA=5;
static const int INT_ROJO=6;
static const int INT_AMARILLO=7;
```

Mostrar el cambio de colores de manera gráfica

Para la lectura de los archivos que se modifican con el objetivo de cambiar colores gráficamente, se probó de muchas maneras que al final no funcionaron correctamente. Se probó con otro proceso pero resulta que no pinta la interfaz gráfica en sí ya que no cuentan con memoria compartida. Entonces al final se decidió usar un Thread con la tarea de leer y pintar los componentes deseados. El run() de este thread se ve algo así:

```
void ThreadLeer::run(){
     qDebug("Run");
    bool aux=true;
    qDebug("Buscando Planta%d\n",planta);
    while (true) {
        qDebug("Buscando Planta%d\n",planta);
        while ( planta!=0) {
            qDebug("Color");
            string myText="";
            std::ostringstream s;
             s<< "Planta"<<planta<<"/color";
             std:: string plantaFileColor(s.str());

            // Read from the text file
            ifstream MyReadFile;
                    MyReadFile.open(plantaFileColor);
            if(MyReadFile){
                while (getline (MyReadFile, myText)) {
                  // Output the text from the file
                  cout << myText;
                }
                if(myText.length()>0){
                    stringstream geek(myText);
                    int x = 0;
                        geek >> x;
                        qDebug("Color--");
                    emit NuevoColor(x,0,0);
                }

            }
```

En este código se implementa un while que verifica que planta se desea observar y otro while que constantemente busca el archivo deseado para modificar el color. El NuevoColor tendrá tres parámetros: el primero es el del código del color, el segundo el número de la rama y el ultimo el numero de la hoja. Según el código del color que se obtiene de archivo este se enviará como un signal para ser recibido con un slot de la siguiente manera.

```
connect(threadLeer,SIGNAL(NuevoColor(int,int,int)),this,SLOT(onNuevoColor(int,int,int)));
```

```
void MainWindow::onNuevoColor(int color,int rama,int hoja){
    qDebug("Entrando %d\n",color);
    string nombreColor = convertint(color);

    if(hoja==0){
        if(rama==0){
            int n2 = nombreColor.length();
                    char colorChar[n2 + 1];
                    strcpy(colorChar, nombreColor.c_str());
                    ui->tronco->setBackground(QBrush(QColor(colorChar)));
        }
    }
```

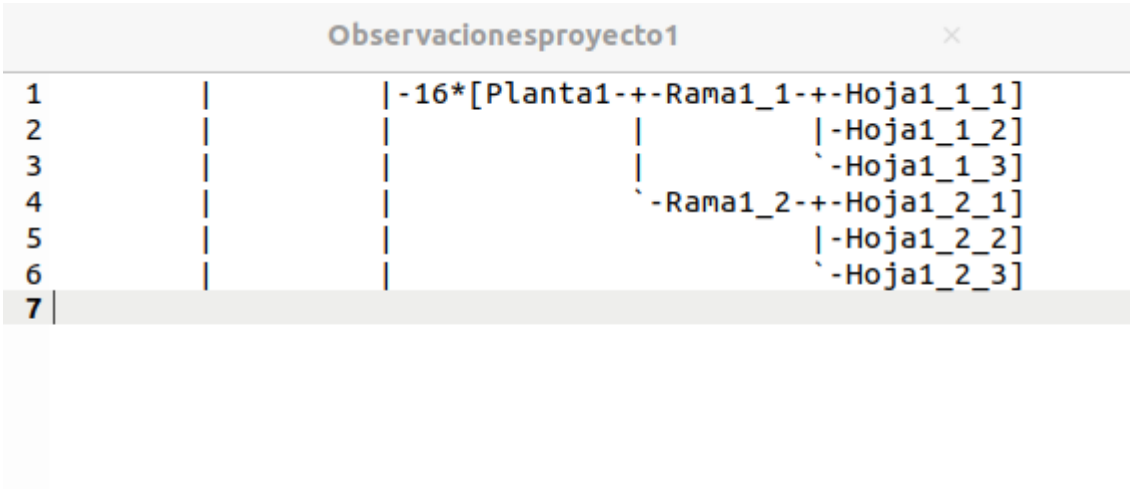
En donde se convertira el int en un string y así pintar de diferente color el componente.

Imprimir el estado del proceso en un archivo

Al momento de obtener un I en el comando, el programa se dirige a la funcion de imprimir esto se verá de la siguiente manera.

```
  ofstream file;
  std::ostringstream s;
  s<< "ImprimirPlanta"<<id;

  char input[50];
          sprintf(input,"pstree |grep Hoja%d",id);


  std::array<char, 128> buffer;
      std::string result;
  std::unique_ptr<FILE, decltype(&pclose)> pipe(popen(input, "r"), pclose);
      if (!pipe) {
          throw std::runtime_error("popen() failed!");
      }
      while (fgets(buffer.data(), buffer.size(), pipe.get()) != nullptr) {
          result += buffer.data();
      }



  file.open(s.str());
  file <<result ;
  file.close();
```

En donde se usura popen() para ejecutar un comando y guardar la salida. Este comando será el póster con un grep del id de la planta que se desea ver. Esta salida se guardará en el archivo que se abrió con file y finalmente se cerrará el buffer. El archivo de imprimir se vera algo asi:



RESULTADOS

Todos los procesos del árbol hicieron el trabajo indicado y esto llevó a los siguientes resultados.

Creación de los procesos(vídeo)

En este primer video se observa cómo se crean los procesos, además de la función del imprimir. Con el pstree se ven los procesos y también los archivos que van modificando

Cambio de color(vídeo)

Aqui se observa como cambia de color el tronco del arbol segun lo que indique el proceso.

Cambio de color2 (vídeo)

Aqui se observa como cambian de color el tronco, ramas y hojas del arbol segun lo que indique el proceso.

https://github.com/SergioCifuentes/TreeOfProcesses.git

CONCLUSIÓN

Los procesos de linux en ocasiones pueden llegar a ser complicados e interesantes. Pero es importante saber manejarlos para una mayor comprensión del sistema operativo. El sistema multitareas de algunos OS nos abren la posibilidad de lograr muchas cosas pero estas no se podran si tener en claro el concepto de procesos padres e hijos, además del entendimiento de las muchas formas de comunicación entre ellos.

BIBLIOGRAFÍA

http://sopa.dis.ulpgc.es/ii-dso/leclinux/procesos/fork/LEC7_FORK.pdf

https://www.ochobitshacenunbyte.com/2018/06/04/visualizar-arbol-de-procesos-en-linux-con-pstree/

https://doc.qt.io/qt-5/qtwidgets-tutorials-addressbook-part1-example.html

https://doc.qt.io/qt-5/signalsandslots.html

https://stackoverflow.com/questions/478898/how-do-i-execute-a-command-and-get-the-output-of-the-command-within-c-using-po

https://www.tutorialspoint.com/the-best-way-to-check-if-a-file-exists-using-standard-c-cplusplus