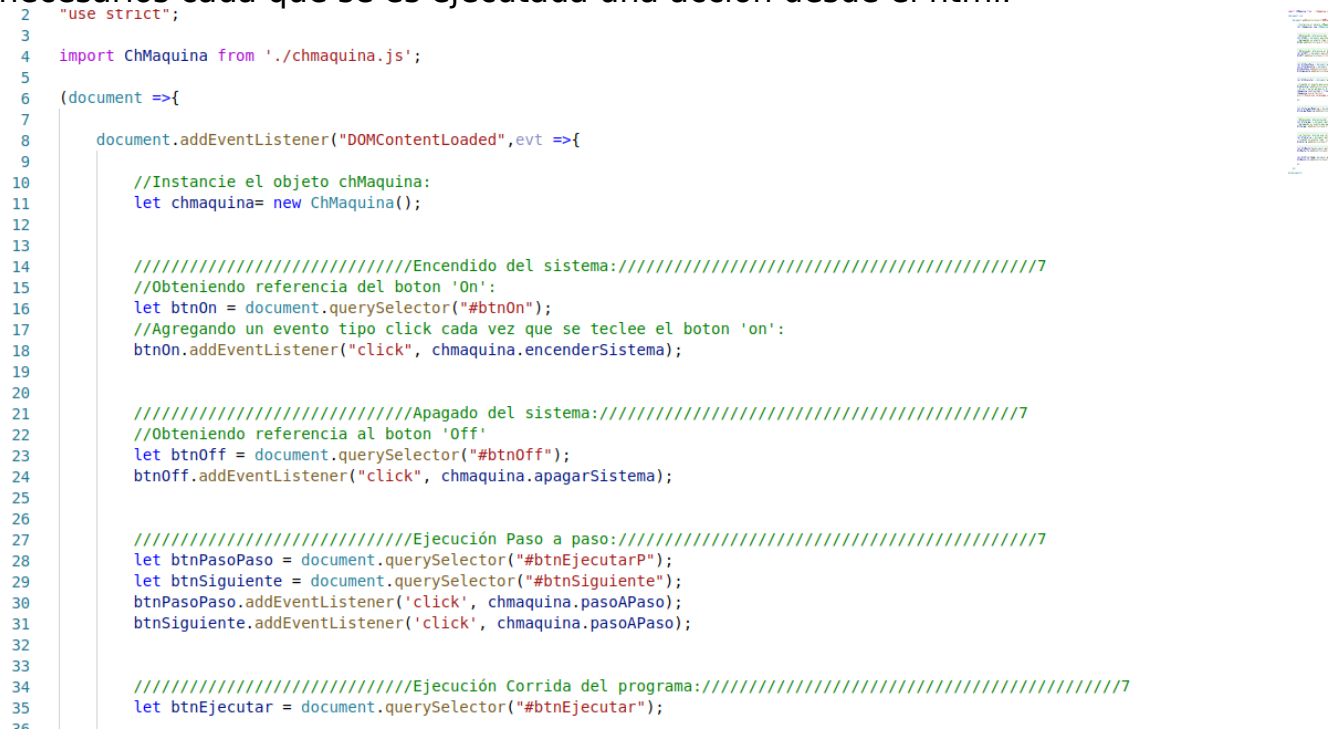
Manual técnico Ch- Maquina Fase 2.

¡Nuevos cambios surgen! Haciendo énfasis en las buenas practicas de programación se han implementado cambios importantes a la estructura de el software orientando el proyecto a objetos dejando como estructura lógica los siguientes componentes:

main.js: Se trata de el módulo principal que hace a su vez de intermediario entre el usuario (capa html) y las clases ‘ch-maquina’ y ‘proceso’. En este módulo se instancia el objeto de la clase ‘ch-maquina’ y se ha programado los eventos necesarios cada que se es ejecutada una acción desde el html.



```
"use strict";

import ChMaquina from './chmaquina.js';

(document =>{

    document.addEventListener("DOMContentLoaded",evt =>{

        //Instancie el objeto chMaquina:
        let chmaquina= new ChMaquina();


        ////////////////////////////////Encendido del sistema:////////////////////////////////////////////////7
        //Obteniendo referencia del boton 'On':
        let btnOn = document.querySelector("#btnOn");
        //Agregando un evento tipo click cada vez que se teclee el boton 'on':
        btnOn.addEventListener("click", chmaquina.encenderSistema);


        ////////////////////////////////Apagado del sistema:////////////////////////////////////////////////7
        //Obteniendo referencia al boton 'Off'
        let btnOff = document.querySelector("#btnOff");
        btnOff.addEventListener("click", chmaquina.apagarSistema);


        ////////////////////////////////Ejecución Paso a paso:////////////////////////////////////////////////7
        let btnPasoPaso = document.querySelector("#btnEjecutarP");
        let btnSiguiente = document.querySelector("#btnSiguiente");
        btnPasoPaso.addEventListener('click', chmaquina.pasoAPaso);
        btnSiguiente.addEventListener('click', chmaquina.pasoAPaso);


        ////////////////////////////////Ejecución Corrida del programa:////////////////////////////////////////////////7
        let btnEjecutar = document.querySelector("#btnEjecutar");
```

chmaquina.js: Clase principal que contiene toda la lógica de nuestro proyecto, aquí se han programado todos los métodos necesarios para la implementación de las fases uno y dos, el analizador léxico, los métodos de planificación de procesos, la ejecución de los procesos y demás funcionalidades programadas que dan vida al ch-maquina.

index.html | main.js | proceso.js | chmaquina.js

```
"use strict";

import Proceso from './proceso.js';

export default class ChMaquina {

    constructor(){

        this.planificacion=6; //Esta variable me permite indicar el metodo de planificacion de procesos, por defecto Round-Robin
        this.diccionarioDirecciones = {}; /*Este diccionario me permitira obtener la ubicacion de una variable en el arreglo de mem
        this.acumulador = 0; /*La variable this.acumulador de vital importancia para la realizacion de las distintas operacionesen
        this.memoria_Principal = []; /*Se crea un vector cuya funcion sera la de almacenar las this.instrucciones de nuestro lengua
        this.kernelSistema; /*Variable que contendra el sistema operativo, por defecto si no se ingresa un valor en kernel este ser
                (10*z+9) el valor minimo del kernel es de 10 posiciones*/
        this.diccionarioEtiquetas = {}; //Este diccionario permite asociar una etiqueta creada por el usuario y una posicion en mem
        //Esta variable me permite llevar una cuenta y conocer que tanto se ha usado de la memoria
        this.contadorMemoria = 0;
        //Este vector es muy importante pues dentro almacena cada uno de los vectores que a su vez contienen cada una de las lineas
        this.vectorInstrucciones = [];

        /*Esta variable determina si el sistema se encuentra this.encendido o this.apagado--Por defecto se encuentra en OFF*/
        this.encendido = false;
        this.apagado = true;

        //Esta variable permite alternar entre ejecutar las this.instrucciones y cargar las this.instrucciones en memoria:
        this.instruccion = false;

        //Variable impreiscindible, contiene en su interior la lista de los programas en memoria(posicion donde empieza, posicion c
        this.listaProgramas = [];

        //Contiene la informacion del programa que actualimente se esta ejecutando:
        this.infoPrograma = [];

        this.copiaInstrucciones = [];
```

Proceso.js: Clase útil que almacena información importante para la ejecución de cada proceso, esta clase almacena atributos tales como el valor actual de el acumulador, una copia con las instrucciones que se están ejecutando hasta ese momento, la longitud de el proceso, la linea de ejecución del programa antes de este ser expropiado entre otros atributos relevantes.

index.html | main.js | proceso.js | chmaquina.js

```
"use strict";

export default class Proceso{

    constructor(infoPrograma){
        console.log("hola");
        //la lista almacena alguna informacion del programa.
        this.infoPrograma=infoPrograma;
        //Linea donde ha quedado el programa en ejecución antes de ser expropiado.
        this.lineaEjecucion=0;
        //Valor actual del acumulador al momento de ser expropiado
        this.acumulador_actual;
        //vector que contiene las instrucciones de el procesito para ser ejecutadas.
        this.copiaInstrucciones=[];
        this.longitudPrograma=0;
        //Este array contiene la información de copiaInstrucciones, pero ya lista para ser ejecutada
        this.arrayFormateado=[];



    }
}
```

Planificación De Procesos.

Dado que nuestro ch-maquina no debe permitir el cambio de planificación de procesos en ejecución se ha decidido que el usuario pueda escoger una opción unicamente cuando la ch-maquina se encuentra apagada.

La opción escogida es almacenada en una variable y comparada en un 'switch-case' para posteriormente tomar decisión sobre el método de planificación seleccionado.

```
if (this.encendido == true && this.apagado == false) {
    var n = document.getElementById('txtComandos').value
    console.log(n);
    document.getElementById("lblAvisoModo").textContent = "Modo usuario";
    //Como el primero que entra es el primero en ser atendido se extrae el programa que primero ingreso
    //Si lista de programas es mayor a cero es porque hay ya un programa cargado en memoria:
    if (this.instruccion == true && this.listaProgramas.length > 0) {
        switch(this.planificacion){

            case "1":
                this.fcfs();
                break;

            case "2":
                this.ejecucionPrioridadNoExpropiativo(this.listaProgramas);

            case "3":
                break;

            case "4":
                this.ejercutarMasCorto(this.listaProgramas);
                break;

            case "5":
                break;
            case "6":
                this.ejecutarRoundRobin(this.listaProgramas);

                break;

            default:
                break;
        }
```

Planificación de Procesos SJF.

El planificador extrae de la lista circular el proceso que se encuentra en primer posición para iniciar la comparación y con un bucle 'for' se recorre la lista comparando dicho proceso con cada uno de los ítems de la lista, si encuentra uno con longitud menor de instrucciones lo asigna como menor hasta encontrar el proceso con menor longitud.

A continuación se consulta la posición de dicho proceso en la lista y se extrae, una vez extraído se obtiene su secuencia de instrucciones, se formatea y se deja listo para ser ejecutado.

```
/**
 * Realiza una ejecución corrida de la lista de programas usando SJF
 * @param:{listaProgramas}: lista circular que contiene cada proceso almacenado en memoria.
 */
ejercutarMasCorto(listaProgramas){
    this.infoPrograma=[];
    this.acumulador=0;
    //Obteniendo la longitud del programa:
    let menor=listaProgramas[0][2].length;
    let programaElegido=listaProgramas[0];

    for(let prog of listaProgramas){
        if(prog[2].length<=menor){
            programaElegido=prog;
            menor=prog[2].length;
        }

    }

    //Obteniendo la posición del programa elegido:
    let posicion=listaProgramas.indexOf(programaElegido);
    //Se extrae el proceso con menor linea de instrucciones:
    this.infoPrograma= listaProgramas.slice(posicion)[0];
    this.vectorInstrucciones=this.infoPrograma[2];
    //El metodo formatea los comandos en array entendibles para ser ejecutados
    this.convertirVectorInstrucciones();
    listaProgramas.splice(posicion,1);

    this.instruccion=true;

    this.copiaInstrucciones = this.infoPrograma[2].slice();
    let contador=this.copiaInstrucciones.length;
    this.copiaInstrucciones = this.formatearCadena(this.copiaInstrucciones);


    this.limpiarInterfaz();
    for(let i=0;i<contador; i++){
        let comando = this.quitarEspacios(this.copiaInstrucciones);
        this.ejecutarInstrucciones(comando, 1);

    }

    this.eliminarTabla();
    this.generarTabla(false);

}
```

Ejecución por prioridades.

La lógica asumida es muy similar a la anterior, se parte desde un proceso inicial asumiendo que esa es la mayor prioridad, cuando el bucle recorre la lista circular compara cada uno de los elementos obteniendo al final la mejor prioridad, se busca y se extrae de la lista dicho proceso y se formatean los comandos para su ejecución.

```
/**
 * Realiza una ejecución corrida de la lista de programas usando prioridad no expropiativa
 * @param:{listaProgramas}: lista circular que contiene cada proceso almacenado en memoria.
 */
ejecucionPrioridadNoExpropiativo=(listaProgramas)=>{
    this.infoPrograma=[];
    this.acumulador=0;
    //Obteniendo la prioridad del programa:
    let mayor=listaProgramas[0][6];
    let programaElegido=listaProgramas[0];

    for(let prog of listaProgramas){
        if(prog[6]>=mayor){
            programaElegido=prog;
            mayor=prog[6];
        }

    }

    //Obteniendo la posición del programa elegido:
    let posicion=listaProgramas.indexOf(programaElegido);
    //Se extrae el proceso con mayor prioridad:
    this.infoPrograma= listaProgramas.slice(posicion)[0];
    this.vectorInstrucciones=this.infoPrograma[2];
    //El metodo formatea los comandos en array entendibles para ser ejecutados
    this.convertirVectorInstrucciones();
    listaProgramas.splice(posicion,1);

    this.instruccion=true;

    this.copiaInstrucciones = this.infoPrograma[2].slice();
    let contador=this.copiaInstrucciones.length;
    this.copiaInstrucciones = this.formatearCadena(this.copiaInstrucciones);


    this.limpiarInterfaz();
    for(let i=0;i<contador; i++){
        let comando = this.quitarEspacios(this.copiaInstrucciones);
        this.ejecutarInstrucciones(comando, 1);

    }

    this.eliminarTabla();
    this.generarTabla(false);

};
```

Nota: Cuando los procesos pasan por cada uno de los métodos de planificación su sintaxis ya ha sido previamente analizada y se asume que no hay errores de este tipo, si un proceso contiene errores de programación en su sintaxis nunca llegará a esta instancia.

Planificación Round Robin.

Aquí el problema se torna un poco complejo, para facilitar las cosas hemos creado la clase ‘Proceso’ la cual nos permite cambiar de contexto cada que un proceso

necesita ser expropiado. Si el programa se inicia en esta opción preguntará por el quantum que se desea manejar. Cada que un proceso es ejecutado por primera vez se instancia la clase proceso y este a su vez es almacenado en una lista de procesos.

```
}

ejecutarRoundRobin=(listaProgramas)=>{
    //La lista Round robin contiene el estado de los procesos en cola.
    this.listaRoundRobin=[];
    this.acumulador=0;

    //Extraiga el primer procesito:
    let procesoElegido=new Proceso(listaProgramas.shift());

    this.vectorInstrucciones=procesoElegido.infoPrograma[2];
    //El metodo formatea los comandos en array entendibles para ser ejecutados
    this.convertirVectorInstrucciones();

    this.instruccion=true;

    //Asigna a el objeto proceso una copia de los comandos:
    procesoElegido.copiaInstrucciones = this.infoPrograma[2].slice();
    //Cantidad de lineas de instrucción que contiene el proceso:
    procesoElegido.longitudPrograma=procesoElegido.copiaInstrucciones.length;
    this.copiaInstrucciones=procesoElegido.copiaInstrucciones;
    //Obtiene un array formateado listo para ser ejecutado:
    this.copiaInstrucciones=this.formatearCadena(this.copiaInstrucciones);
    procesoElegido.arrayFormateado=this.copiaInstrucciones.slice();
    this.listaRoundRobin.push(procesoElegido);
    let contador=procesoElegido.arrayFormateado.length;
    this.limpiarInterfaz();
    for(let i=0;i<contador-1; i++){
        let comando = this.quitarEspacios(this.copiaInstrucciones);
        if(comando!=''){
            this.ejecutarInstrucciones(comando, 1);

        }
```