Manual técnico CH-Máquina

Luis Felipe Vélez García

Universidad Nacional de Colombia sede Manizales

Administración de sistemas informáticos

Sistemas operativos

Manizales, Caldas.

2023

Introducción

El CH-Máquina es un software de simulación de sistema operativo que funciona en computadoras. Permite la ejecución de programas con la extensión .CH, los cuales deben estar escritos en un lenguaje específico llamado CH-Lenguaje. Estos programas siguen operaciones básicas comunes en otros lenguajes de programación. La característica principal del software es la gestión de memoria, que opera de manera similar a la memoria en un sistema operativo real. Además, el programa puede validar la sintaxis de los programas escritos en CH-Lenguaje para evitar errores y mejorar la eficiencia. Esta validación sintáctica asegura que los programas se ejecuten sin problemas y produzcan los resultados deseados.

La ejecución de programas en la CH-Máquina es secuencial, lo que significa que procesa las instrucciones en el orden en que se han escrito. Este enfoque garantiza que las tareas se realicen en el momento y lugar adecuado, siguiendo un proceso fundamental presente en la mayoría de los lenguajes de programación.

El CH-Máquina fue desarrollado utilizando la plataforma de programación Visual Studio Code y se empleó una combinación de lenguajes de programación HTML, CSS y Javascript. Esta combinación posibilitó la creación de una interfaz de usuario atractiva y funcional, así como la implementación de todas las funcionalidades del software. HTML y CSS se utilizaron para estructurar y diseñar la interfaz gráfica del usuario, mientras que Javascript se utilizó para la lógica del programa y para conectar los diferentes elementos de la interfaz con el código fuente. La elección de estos lenguajes fue crucial para el desarrollo exitoso del CHMáquina y permitió crear un software de alta calidad y fácil de usar para los usuarios.

Para ejecutar el CH-Máquina, primero es necesario abrir el archivo CHMaquina.html en un navegador web compatible. Este archivo se encuentra en la ubicación donde se ha descargado el software. Una vez que el archivo CHMaquina.html se ha abierto en el navegador, el software debe cargarse en la página web para que esté disponible para su uso. Una vez cargado, el usuario puede comenzar a utilizarlo para escribir, cargar y ejecutar programas en el lenguaje CH.

En el desarrollo del software, se utilizó Materialize como framework de diseño para construir la interfaz gráfica de usuario. Materialize es un framework de front-end basado en CSS y JavaScript que facilita el diseño de páginas web responsivas con un aspecto visualmente atractivo y moderno. Proporciona una variedad de **componentes y estilos** predefinidos que se pueden aprovechar para crear una interfaz de usuario coherente y fácil de usar.

Html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css" rel="stylesheet"
integrity="sha384-T3c6CoIi6uLrA9TneNEoa7RxnatzjcDSCmG1MXxSR1GAsXEV/Dwwykc2MPK8M2HN" crossorigin="anonymous">
    <link rel="stylesheet" href="
https://cdnjs.cloudflare.com/ajax/libs/materialize/1.0.0/css/materialize.min.css">
    <script src="https://cdnjs.cloudflare.com/ajax/libs/materialize/1.0.0/js/materialize.min.js"></script>
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <link rel="stylesheet" href="style.css">
    <link href="https://fonts.googleapis.com/icon?family=Material+Icons" rel="stylesheet">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>CHMáquina</title>
</head>
<body>
  <nav class="nav-extended white">
    <div class="nav-content grey">
      <div class="nav-wrapper">
      </div>
      <a href="#" class="brand-logo right" id="navText">CHMÁQUINA Luis Vélez</a>
      <ul class="tabs tabs-transparent">
        <li class="tab">
          <div class="file-field input-field">
            <a class="btn black" id="btnCargar">
              <span >Cargar</span>
              <input type="file" id="fileInput">
              <i class="material-icons left">file_upload</i>
            </a>
          </div>
        </li>
        <li class="tab" ><a class="waves-effect waves-light black btn" id="btnAuto"><i class="
material-icons left">play_arrow</i>Ejecutar</a></li>
        <li class="tab"><a class="waves-effect waves-light black btn" id="btnPasoAPaso"><i class="
material-icons left">directions_walk</i>Paso a paso</a></li>
        <li class="tab"><a class="waves-effect waves-light black btn" id="btnPausar"><i class="
material-icons left">pause</i>Pausa</a></li>
        <li class="tab"><a class="waves-effect waves-light black btn" onclick="mostrarMemoria()"><i class="
material-icons left">list</i>Muestra de memoria</a></li>
      </ul>
    </div>
  </nav>
```

```
<div class="imagenes">
    <img src="imagenes/Procesador.png" alt="CHM-CPU" id="cpuImagen">
    <img src="imagenes/computador.png" alt="CHM-CPU" id="monitorImagen">
    <img src="imagenes/impresora.png" alt="CHM-CPU" id="impresoraImagen">
  </div>
  <div class="row">
    <div class="col s1">
      <h6 id="textProgramMode" class="center">Modo <span id="programMode"></span></h6>
      <h6 id="nameProgram" class="center"><span id="programName"></span></h6>
      <div class="col s3.8 white-text">
        <form action="#">
          <p class="range-field">
            <h6 id="text">Memoria : <span id="memoryValor"></span></h6>
            <input type="range" id="memoryRange" min="3" max="7100" value="120" />
          </p>
        </form>
        <form action="#">
          <p class="range-field">
            <h6 id="text">Kernel : <span id="kernelValor"></span></h6>
            <input type="range" id="kernelRange" min="1" max="7098" value="79" />
          </p>
        </form>
        <h6 id="text">Acumulador: <span id="acumuladorValor"></span></h6>
        <h6 id="text">Instrucción: <span id="instructionLine"></span></h6>
      </div>
      <div class="card left" id="cardScreen">
        <div class="card-content">
          <p id="screen" class="white-text"></p>
        </div>
      </div>
      <div class="card" id="cardImpresora">
        <div class="card-content">
          <p id="print" class="black-text"></p>
        </div>
      </div>
    </div>
    <div class="col s5">
      <div class="col s2">
        <div class="card" id="cardFile">
          <div class="card-content black-text ">
            CÓDIGO<p id="readFile" class="black-text"></p>
          </div>
        </div>
      </div>
      <div class="col s2">
        <div class="card" id="cardVariables">
          <div class="card-content black-text">
            VARIABLES<br><p id="readVariables" class="black-text"></p>
          </div>
        </div>
        <div class="card" id="cardEtiquetas">
          <div class="card-content black-text ">
            ETIQUETAS<br><p id="readEtiquetas" class="black-text"></p>
          </div>
        </div>
      </div>
    </div>
```

```
<div class="col s2" >
      <div class="card" id="cardMuestraMemoria">
        <div class="card-content black-text">
          MEMORIA<br><p id="readMuestraMemoria" class="black-text"></p>
        </div>
      </div>
    </div>
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/js/bootstrap.bundle.min.js" integrity="
sha384-C6RzsynM9kWDrMNeT87bh95OGNyZPhcTNXj1NW7RuBCsyN/o0jlpcV8Qyq46cDfL" crossorigin="anonymous"></script>
  <script src="CHM.js"></script>
</body>
</html>
```

JavaScript

Variables globales

Variables globales declaradas para el uso lo largo de todo el programa.

```
M.AutoInit();

//lista de nemónicos para el procesador
//la función "reinicio" asigna 0 si es entero, flotante o booleano ó asigna "" si es una cadena, a la variable
señalada por el operando
var lista= ["cargue","almacene","nueva","lea","sume","reste","multiplique","divida","potencia","modulo","
concatene","elimine","extraiga","Y","O","NO","muestre","imprima","retorne\r","retorne","vaya","vayasi","
etiqueta","reinicio"];
var memoria = new Array ();
var asignado=false;
var sizeDocumentos=0;
var primerDocumento = true;
var acumulador = 0;
var idDocumento = 0;
var diccionarioProcesos = {};
var idProceso = 1;
var index = 0;
var intervaloID = 0;
var modoPasoAPaso = false;
var documento=[];
var URLactual=[];
var proceso=[];
var contador=0;
var pasoapaso=false;
var indicador=1;
```

Botones asignados

Esta es la asignación de los botones que visibles para el usuario y son los que permiten la interacción con el CH-Máquina, se asignan a variable para luego ser usadas durante la operación del programa. Los botones “Slider” se utilizan para conocer el espacio asignado por el usuario a la memoria y al kernel.

```
//Rango de memoria y valor actual
const sliderM = document.getElementById("memoryRange");
const valorM = document.getElementById("memoryValor");
//Rango de kernel y valor actual
const sliderK = document.getElementById("kernelRange");
const valorK = document.getElementById("kernelValor");
//modo del programa
document.getElementById("programMode").innerHTML = "Kernel"
// asignación de botones y funcion para ejecutar el programa en los dos modos diferentes
const botonEjecutarAuto=document.getElementById("btnAuto");
const botonEjecutarPasoAPaso=document.getElementById("btnPasoAPaso");
const botonPausar=document.getElementById("btnPausar");
const botonCargar=document.getElementById("btnCargar");
//
botonEjecutarAuto.addEventListener("click", function(){
    if(asignado){
        botonEjecutarAuto.setAttribute("disabled", "true");
        iniciar();
    }

});
botonEjecutarPasoAPaso.addEventListener("click", function(){
    ventanaContinuar();
});
botonPausar.addEventListener("click", function(){
    //activar boton ejecutar
    ventanaPausar();
});

// Mostrar el valor actual del slider cuando se mueve
valorM.innerText = sliderM.value;
sliderM.addEventListener("input", function() {
    valorM.innerText = sliderM.value;
    sliderK.max = sliderM.value-2;
    valorK.innerText = sliderK.value;
});
// Mostrar el valor actual del slider cuando se mueve
valorK.innerText = sliderK.value;
sliderK.addEventListener("input", function() {
    valorK.innerText = sliderK.value;
});

//evento de cargar archivo
const cargar=document.getElementById("fileInput");
```

Funciones

Lectura, validación de sintaxis y asignación de memoria

Función que valida el contenido del documento cargado, realizando una comparación con la lista de operaciones definidas para el programa. El archivo cargado se recorre línea por línea, haciendo una separación por palabras separadas por un espacio, almacenando dicha separación en la variable “words”. Durante la lectura del archivo se valida si pertenece a alguna de las operaciones predefinidas y sí la estructura de los parámetros es correcta, de ser correcta, se almacenan las “variables” y las “etiquetas” en diccionarios, de presentar algún error de sintaxis, se clasifica dicho error, se almacena en un “erroresArray” y se presentan al usuario en una ventana emergente. Durante la lectura se almacena en la variable “sizeDocumento” la información del total de líneas leídas. Si no se encuentran errores durante la lectura se llama a la función “asignarMemoria” con los datos: “diccionarioVariables”,”diccionarioEtiquetas”,”sizeDocumento”.

```
//funcion para leer archivo y validar sintaxis
function leerArchivo(archivo){
    const erroresArray = new Array();
    let sintaxisError=false;
    const diccionarioVariables = {}
    const diccionarioEtiquetas = {}
    let retorne = false;
    let sizeDocumento = 0;
    const apuntadorEtiqueta =[];
    const lineaEtiqueta =[];
    const logicas = [];
    const vaya = {};
    const vayasi = {};
    const cadena = [];
    const entero = [];
    const real = [];
    archivo.onload = () => {
        //separar por lineas y guardar en un array
        const lines = archivo.result.split('\n');
        //recorrer el array de lineas
        for ( let i = 0; i < lines.length; i++) {
            const line = lines[i];
            //separar por espacios y guardar en un array
            const words = line.split(" ");
            //recorrer el array de palabras
            //si la palabra es un nemónico
            if (lista.includes(words[0]) && !retorne){
                //si se está declarando una variable nueva
                if (words[0] == "nueva" && words.length >= 3) {
                    if (words[1] == "acumulador") {
                        erroresArray.push("Error 08 - Nombre reservado - Línea " + (i + 1));
                        sintaxisError = true;
                    } else if (words[1].length < 255 && esLetra(words[1]) && words.length == 4) {
                        if (!(words[1] in diccionarioVariables)) {
                            diccionarioVariables[words[1]] = quitarSlash(words[3]);
                            if (words[2] === "I") {
                                if (!((Number.isInteger(Number.parseInt(words[3]))) && !isNaN(words[3]) && (!
words[3].includes(".")))) {
                                    erroresArray.push("Error 02 - Tipo de valor - Línea " + (i + 1));
                                    sintaxisError = true;
                                }else{
                                    entero.push(words[1]);
                                }
```

```
} else if (words[2] === "R") {
                                if (!(!isNaN(Number.parseFloat(words[3])) && !isNaN(words[3]))) {
                                    erroresArray.push("Error 02 - Tipo de valor - Línea " + (i + 1));
                                    sintaxisError = true;
                                }else{
                                    real.push(words[1]);
                                }
                            } else if (words[2] === "L") {
                                if (words[3] === "1\r" || words[3] === "0\r") {
                                    logicas.push(words[1]);
                                } else {
                                    erroresArray.push("Error 02 - Tipo de valor - Línea " + (i + 1));
                                    sintaxisError = true;
                                }
                            }else if (words[2] === "C") {
                                if (!typeof words[3] === 'string') {
                                    erroresArray.push("Error 02 - Tipo de valor - Línea " + (i + 1));
                                    sintaxisError = true;
                                }else{
                                    cadena.push(words[1]);
                                }
                            }
                        } else {
                            erroresArray.push("Error 03 - Variable ya declarada - Línea " + (i + 1));
                            sintaxisError = true;
                        }
                    }else if(words[1].length < 255 && esLetra(words[1]) && words.length == 3){
                        if (!(words[1] in diccionarioVariables)) {
                            if (words[2] === "I\r") {
                                diccionarioVariables[words[1]] = "0";
                            } else if (words[2] === "R\r") {

                                diccionarioVariables[words[1]] = "0.0";
                            } else if (words[2] === "L\r") {
                                logicas.push(words[1]);
                                diccionarioVariables[words[1]] = "0";
                            }else if (words[2] === "C\r") {
                                diccionarioVariables[words[1]] = " ";
                            }
                        } else {
                            erroresArray.push("Error 03 - Variable ya declarada - Línea " + (i + 1));
                            sintaxisError = true;
                        }
                    }else {
                        erroresArray.push("Error 01 - Error 06 - Sintaxis - Longitud - Línea " + (i + 1));
                        sintaxisError = true;
                    }
```

```
} else if (words[2] === "R") {
                                if (!(!isNaN(Number.parseFloat(words[3])) && !isNaN(words[3]))) {
                                    erroresArray.push("Error 02 - Tipo de valor - Línea " + (i + 1));
                                    sintaxisError = true;
                                }else{
                                    real.push(words[1]);
                                }
                            } else if (words[2] === "L") {
                                if (words[3] === "1\r" || words[3] === "0\r") {
                                    logicas.push(words[1]);
                                } else {
                                    erroresArray.push("Error 02 - Tipo de valor - Línea " + (i + 1));
                                    sintaxisError = true;
                                }
                            }else if (words[2] === "C") {
                                if (!typeof words[3] === 'string') {
                                    erroresArray.push("Error 02 - Tipo de valor - Línea " + (i + 1));
                                    sintaxisError = true;
                                }else{
                                    cadena.push(words[1]);
                                }
                            }
                        } else {
                            erroresArray.push("Error 03 - Variable ya declarada - Línea " + (i + 1));
                            sintaxisError = true;
                        }
                    }else if(words[1].length < 255 && esLetra(words[1]) && words.length == 3){
                        if (!(words[1] in diccionarioVariables)) {
                            if (words[2] === "I\r") {
                                diccionarioVariables[words[1]] = "0";
                            } else if (words[2] === "R\r") {

                                diccionarioVariables[words[1]] = "0.0";
                            } else if (words[2] === "L\r") {
                                logicas.push(words[1]);
                                diccionarioVariables[words[1]] = "0";
                            }else if (words[2] === "C\r") {
                                diccionarioVariables[words[1]] = " ";
                            }
                        } else {
                            erroresArray.push("Error 03 - Variable ya declarada - Línea " + (i + 1));
                            sintaxisError = true;
                        }
                    }else {
                        erroresArray.push("Error 01 - Error 06 - Sintaxis - Longitud - Línea " + (i + 1));
                        sintaxisError = true;
                    }
```

```
}else if(words[0]==="vaya" && (words.length==2)){
                    vaya[quitarSlash(words[1])]=sizeDocumento+1;

                }else if(words[0]==="vayasi" && (words.length==3)){
                    vayasi[quitarSlash(words[1])]=[(quitarSlash(words[2])), sizeDocumento+1];

                }else if((words[0]==="Y" || words[0]==="O") && (words.length==4)){
                    if(!logicas.includes(quitarSlash(words[1])) || (!logicas.includes(quitarSlash(words[2])) ||
 (!logicas.includes(quitarSlash(words[3])))){
                        erroresArray.push("Error 06 - Variable no declarada - Línea " + (i + 1));
                        sintaxisError = true;
                    }
                }else if(words[0]==="NO" && (words.length==3)){
                    if(!logicas.includes(quitarSlash(words[1])) || (!logicas.includes(quitarSlash(words[2
]))){
                        erroresArray.push("Error 06 - Variable no declarada - Línea " + (i + 1));
                        sintaxisError = true;
                    }
                }else if(words[0]==="muestre" && (words.length==2)){
                    if(words[1]!="acumulador\r" && !((quitarSlash(words[1])) in diccionarioVariables)){
                    erroresArray.push("Error 01 - Sintaxis - Línea " + (i + 1));
                    sintaxisError = true;
                    }
                }else if(words[0]==="imprima" && (words.length==2)){
                    if(words[1]!="acumulador\r" && !((quitarSlash(words[1])) in diccionarioVariables)){
                    erroresArray.push("Error 01 - Sintaxis - Línea " + (i + 1));
                    sintaxisError = true;
                    }
                }
            }else if (!((retorne==true)  || words[0] == "\r" || words[0] == "\n" || words[0] == "\r\n" || words
[0] == "\n\r" || words[0] == "" || (words[0].substring(0, 2) === "//"))) {
                //si la palabra es un salto de linea
                erroresArray.push("Error 01 - Sintaxis - Línea " + (i + 1));
                sintaxisError = true;

            }else{
                sizeDocumento--;
            }
            sizeDocumento++;
        }
        //verificar que las etiquetas apunten a una instrucción existente
        Object.entries(diccionarioEtiquetas).forEach(([key, valor]) => {
            if(valor[1]>sizeDocumento){
                erroresArray.push("Error 05 - Posición de etiqueta ("+(key)+")");
                sintaxisError = true;
            }
        });
```

```
//verificar que las etiquetas no apunten a otra etiqueta
        if (apuntadorEtiqueta.some((valor=>lineaEtiqueta.includes(valor)))) {
            erroresArray.push("Error 05 - Etiqueta apuntando a otra etiqueta");
            sintaxisError = true;
        }
        //verificar que la etiqueta indicada en la sentencia vaya exista en el diccionario de etiquetas
        Object.entries(vaya).forEach(([key, valor]) => {
            if(!(key in diccionarioEtiquetas)){
                erroresArray.push("Error 05 - Etiqueta no existente - Línea " + (valor));
                sintaxisError = true;
            }
        });
        //verificar que la etiqueta indicada en la sentencia vayasi exista en el diccionario de etiquetas
        Object.entries(vayasi).forEach(([key, valor]) => {
            if(!(key in diccionarioEtiquetas)){
                erroresArray.push("Error 05 - Etiqueta no existente - Línea " + (valor[1]));
                sintaxisError = true;
            }
            if(!(valor[0] in diccionarioEtiquetas)){
                erroresArray.push("Error 05 - Etiqueta no existente - Línea " + (valor[1]));
                sintaxisError = true;
            }
        });
        //mostrar errores en pantalla emergente
        if (sintaxisError) {
            alert("SE ENCONTRARON ERRORES!!!");
            let ventanaErrores = window.open("", "Errores", "width=400,height=200" );
            ventanaErrores.document.write('<h1>' + "ERRORES: " + "<br>" + '</h1>');
            ventanaErrores.document.body.style.backgroundColor = "black";
            ventanaErrores.document.body.style.color = "white";

// document.getElementById("readFile").insertAdjacentHTML('beforeend', '<p>' + "---------------" + '</p>');
            for (let i = 0; i < erroresArray.length; i++) {

// document.getElementById("readFile").insertAdjacentHTML('beforeend', '<p>' + erroresArray[i] + '</p>');
                ventanaErrores.document.write('<p>' + erroresArray[i] + "<br>" + '</p>');
            }
        } else {
            //si no hay errores, se ejecuta el programa
            const variablesProceso = {};
            //etiquetas
            const etiquetasProceso = {};
            //tamaño de memoria disponible para almacenamiento
            let sizeMemmoriaDisponible=(parseInt(sliderM.value)-((parseInt(sliderK.value)))-1);
            //tamaño del archivo contando el espacio requerido para las variables
            let sizeArchivoYVariables = (sizeDocumento + Object.keys(diccionarioVariables).length);
            sizeDocumentos += sizeArchivoYVariables;
            //tamaño de memoria vs instrucciones
            if (sizeDocumentos > sizeMemmoriaDisponible) {
                alert("EL DOCUMENTO ES MUY GRANDE PARA LA MEMORIA!!!");
                //quitar los valores del archivo a la variable de tamaño general
                sizeDocumentos-=sizeArchivoYVariables;
            }else{
                //desabilitar sliders de rangos
                // sliderM.disabled = true;
                // sliderK.disabled = true;
                // //asignar valores al kernel
                // asignado = true;
                // if(primerDocumento==true){
                //     llenarMemoria();
                //     memoria[0] = acumulador;
                //     for (let i = 1; i <= parseInt(sliderK.value); i++) {
                //         memoria[i] = "--CHSOS V2023--";
                //     }
                //     primerDocumento=false;
                // }
                //leer archivo
```

```
const archivo = new FileReader();
                documento.push(document.getElementById("fileInput").files[0].name);
                //saber la posicion vacia en memoria
                let j=0;
                if(primerDocumento==true){
                    console.log("primer documento");
                    contador = (parseInt(sliderK.value)+1);
                }
                j=contador;

//variable para saber la posicion a la que apunta la etiqueta al sumarla con el valor que esté en el diccionari
o de etiqueta
                let p = contador-1;
                archivo.readAsText(document.getElementById("fileInput").files[0]);
                archivo.onload = function() {
                    //separar por líneas y guardar en un array
                    const lines = archivo.result.split('\n');
                    for ( i = 0 ; i < lines.length; i++) {
                        const line = lines[i];
                        const words = line.split(" ");
                        //mostrar archivo
                        if (!(words[0] == "\r" || words[0] == "\n" || words[0] == "\r\n" || words[0] == "\n\r"
|| words[0] == "" || (words[0].substring(0, 2) === "//"))) {
                            proceso.push(line.toString());
                            memoria[j] = line;
                            //si la palabra es un salto de linea o comentario
                            document.getElementById("readFile").insertAdjacentHTML('beforeend', '<p>'+ (`${j}`.
padStart(4, "0"))+" "+ line + '</p>');
                            j++;
                        }
                    }
                    //mostrar variables en el card
                    Object.entries(diccionarioVariables).forEach(([key, valor]) => {
                        memoria[j] = valor;
                        proceso.push((valor.toString()));
                        variablesProceso[key] = j;
                        document.getElementById("readVariables").insertAdjacentHTML('beforeend', '<p>'+ (`${j}`
.padStart(4, "0"))+" "+(`${idDocumento}`.padStart(4, "0")) + key + '</p>');
                        j++;
                    });
                    //mostrar etiquetas en el card
                    Object.entries(diccionarioEtiquetas).forEach(([key, valor]) => {
                        // posicion de la etiqueta
                        let posicion = p +valor[1]
                        etiquetasProceso[key] = posicion;
                        document.getElementById("readEtiquetas").insertAdjacentHTML('beforeend', '<p>'+ (`${
posicion}`.padStart(4, "0"))+" "+(`${idDocumento}`.padStart(4, "0"))+ key + '</p>');
                    });
                }
```

```
//número de documentos leídos
                idDocumento++;
                //guardar en el diccionario de procesos
                diccionarioProcesos[idDocumento] = [variablesProceso, etiquetasProceso];
                console.log(diccionarioProcesos);
                index=parseInt(sliderK.value)+1;
                sliderM.disabled = true;
                sliderK.disabled = true;
                //asignar valores al kernel
                asignado = true;
                if(primerDocumento==true){
                    llenarMemoria();
                    memoria[0] = acumulador;
                    let j = 0;
                    for (let i = 1; i < memoria.length; i++) {
                        if (i<=parseInt(sliderK.value)){
                            memoria[i] = "--CHSOS V2023--";
                        }else{
                            if (j<proceso.length){
                                console.log(proceso[j]);
                                memoria[i] = proceso[j];
                                j++;
                            }
                        }
                    }
                    indicador=parseInt(sliderK.value)+1;
                    primerDocumento=false;
                }

            }
            contador+=sizeDocumentos;
            sizeDocumento=0;
            sizeDocumentos=0;
        }
    };

}
```

Asignación de memoria:

Seleccionar los valores que serán asignados a la memoria principal del sistema, un vector llamado “memoria”. Primero se valida si el tamaño del archivo cargado no sobre pasa el tamaño de memoria asignado. Luego, asigna las posiciones indicadas para Kernel, utilizando los botones “slider”, durante este proceso se llama otra función de nombre “llenarMemoria”. Después, se lee el archivo nuevamente para asignar línea por línea a los espacios de memoria indicados y realizando la clasificación de las variables y etiquetas. Esta información es presentada en pantalla. En esta función se recolecta la información de la cantidad de documentos leídos en la variable global “idDocumento” y el tamaño de espacio asignado para kernel en la variable global “index”.

Mostrar memoria

La función “mostrarMemoria” es la encargada de mostrar en pantalla el contenido asignado en la memoria.

```
//funcion para mostrar la memoria
function mostrarMemoria() {
    //limpiar memoria
    document.getElementById("readMuestraMemoria").innerHTML = "";
    //validar si la memoria ya fue asignada
    if (asignado == false){
        memoria.length = 0;
        llenarMemoria();
    }
    //mostrar memoria
    for (let i = 0; i < memoria.length; i++) {
        if(i==0){
            document.getElementById("readMuestraMemoria").insertAdjacentHTML('beforeend', '
<p><i class="tiny material-icons">book</i>'+(`${i}`.padStart(4, "0")) +" "+ memoria[i] + '</p>');
        }else if(i<=sliderK.value){
            document.getElementById("readMuestraMemoria").insertAdjacentHTML('beforeend', '
<p><i class="tiny material-icons">filter_tilt_shift</i>'+(`${i}`.padStart(4, "0")) +" "+ memoria[i] + '</p>');
        }else{
            document.getElementById("readMuestraMemoria").insertAdjacentHTML('beforeend', '
<p><i class="tiny material-icons">memory</i>'+ (`${i}`.padStart(4, "0")) +" "+ memoria[i] + '</p>');
        }
    }
}
```

Llenar Memoria

La función “llenarMemoria” asigna vacío en las posiciones de memoria indicadas por el usuario con el fin de ser modificadas más adelante.

```
//funcion para llenar la memoria con vacios
function llenarMemoria(){
    for (let i = 0; i < (parseInt(sliderM.value)); i++) {
        memoria[i] ="";
    }
}
```

Vacío

La función “esVacio” se encarga de retornar la primera posición vacía dentro de la memoria.

```
//funcion para saber la posicion vacia de la memoria
function esVacio(){
    let posicionVacia = 0;
    for (let i = 1; i < memoria.length; i++) {
        if (memoria[i] == "") {
            posicionVacia = i;
            break;
        }
    }
    return posicionVacia;
}
```

Letra

La función “esLetra” recibe un parámetro y retorna si el primer carácter es una letra o no.

```
//funcion para validar si es una letra
function esLetra(cadena) {
    let primerCaracter = cadena.charAt(0);
    let codigoUnicode = primerCaracter.charCodeAt(0);
    if ((codigoUnicode >= 65 && codigoUnicode <= 90) || (codigoUnicode >= 97 && codigoUnicode <= 122)) {
      return true; // Si el primer carácter es una letra mayúscula o minúscula, devuelve true
    } else {
      return false; // Si el primer carácter no es una letra devuelve false
    }
}
```

Remover Slash

La función “quitarSlash” recibe un parámetro, lo separa, quitando así el "\r" y lo retorna.

```
//funcion para quitar el contra slash
function quitarSlash(cadena){
    let cadenaSinSlash = cadena.split("\r");
    return cadenaSinSlash[0];
}
```

Ejecutar

Esta función recibe como parámetro la variable “index”. Valida si el contenido de la posición de memoria es “retorne” y realiza el llamado a la función “accion”, modificando el valor de la variable “index”, acá llamada “i”. También, muestra en pantalla el contenido de

la posición de la memoria y el valor de la posición 0 “acumulador”. Esta función retorna el valor de “i”.

```
//funcion para ejecutar las instrucciones de la memoria
function ejecutar(i){
    //mostrar nombre de programa en el card
    if(documento.length >= idProceso){
        document.getElementById("programName").innerHTML= documento[idProceso-1];
    }
    try{
        try{
            let line = memoria[i].split(" ");
            if ((line.length >= 2)){
                if (line[0] === "retorne" || line[0] === "retorne\r"){
                    idProceso++;
                    memoria[0] = 0;
                    //salto de linea
                    document.getElementById("screen").innerHTML+= "<br>";
                }
                document.getElementById("instructionLine").innerHTML = (`${i}`.padStart(4, "0")+"-"+ memoria[i
]);
                //mostrar acumulador en el card
                document.getElementById("acumuladorValor").innerHTML = memoria[0];
                //ejecutar la instruccion
                i=accion(memoria[i],idProceso,i);
            }else if(line[0]==="retorne" || line[0]==="retorne\r"){
                idProceso++;
                memoria[0] = 0;
                document.getElementById("instructionLine").innerHTML = (`${i}`.padStart(4, "0")+"-"+ memoria[i
]);
                //mostrar acumulador en el card
                document.getElementById("acumuladorValor").innerHTML = memoria[0];
                //salto de linea
                document.getElementById("screen").innerHTML+= "<br>";

            }
        }catch(error){
            i=accion(memoria[i],idProceso,i);
        }
    }catch(error){
        console.log("Error en la acción");
    }
    return i;
}
```

Acción

Esta función recibe los parámetros “línea”,” idDocumento” y “i”. Es la encargada de realizar la acción adecuada dependiendo del valor leído en la posición de memoria actual. Realiza una separación del contenido de la posición de la memoria, compara la palabra predefinida y hace la operación adecuada para cada caso. Retorna el valor de “i”.

```
//funcion para ejecutar las acciones de las instrucciones
function accion(linea, idDocumento,i){
    let variable = "";
    let posicion = 0;
    try{
        let operacion = linea.split(" ");
        variable = quitarSlash(operacion[1]);
        posicion = diccionarioProcesos[idDocumento][0][variable];
    }catch(error){
        return i;
    }
    let operacion = linea.split(" ");
    if (operacion[0]==="cargue"){
        //cargue
        memoria[0] = memoria[posicion];
        return i;
    }else if(operacion[0]==="almacene"){
        //almacene
        memoria[posicion] = memoria[0];
        return i;
    }else if(operacion[0]==="lea"){
        //lea
        memoria[posicion] = prompt("Ingrese un valor");
        return i;
    }else if(operacion[0]==="sume"){
        //sume
        let cadena = memoria[posicion].toString();
        let cadena2 = memoria[0].toString();
        if (cadena.includes(".") || cadena2.includes(".")){
            memoria[0] = parseFloat(memoria[0]) + parseFloat(memoria[posicion]);
        }else{
            memoria[0] = parseInt(memoria[0]) + parseInt(memoria[posicion]);
        }
        return i;
    }else if(operacion[0]==="reste"){
        //reste
        let cadena = memoria[posicion].toString();
        let cadena2 = memoria[0].toString();
        if (cadena.includes(".") || cadena2.includes(".")){
            memoria[0] = parseFloat(memoria[0]) - parseFloat(memoria[posicion]);
        }else{
            memoria[0] = parseInt(memoria[0]) - parseInt(memoria[posicion]);
        }
        return i;
    }else if(operacion[0]==="multiplique"){
        //multiplique
        let cadena = memoria[posicion].toString();
        let cadena2 = memoria[0].toString();
        if (cadena.includes(".") || cadena2.includes(".")){
            memoria[0] = parseFloat(memoria[0]) * parseFloat(memoria[posicion]);
        }else{
            memoria[0] = parseInt(memoria[0]) * parseInt(memoria[posicion]);
        }
        return i;
    }else if(operacion[0]==="divida"){
        //divida
        //validar que no se divida entre 0
        try {
            if(memoria[posicion]!= 0){
                let cadena = memoria[posicion].toString();
                let cadena2 = memoria[0].toString();
                if (cadena.includes(".") || cadena2.includes(".")){
                    memoria[0] = parseFloat(memoria[0]) / parseFloat(memoria[posicion]);
                }else{
                    memoria[0] = parseInt(memoria[0]) / parseInt(memoria[posicion]);
                }
            }
        } catch (error) {
            console.log("No se puede dividir entre 0");
        }
        return i;
```

```
}else if(operacion[0]==="potencia"){
        //potencia
        memoria[0] = Math.pow(memoria[0],memoria[posicion]);
        return i;
    }else if(operacion[0]==="modulo"){
        //modulo
        memoria[0] = parseInt(memoria[0]) % parseInt(memoria[posicion]);
        return i;
    }else if(operacion[0]==="concatene"){
        //concatene
        memoria[0] = memoria[0] + memoria[posicion];
        return i;
    }else if(operacion[0]==="elimine"){
        //elimine
        memoria[0] = memoria[0].replaceAll(memoria[posicion],"");
        return i;
    }else if(operacion[0] === "extraiga"){
        //extraiga
        let extraer = memoria[diccionarioProcesos[idDocumento][0][variable]];
        let subcadena = memoria[0].substring(0,extraer);
        memoria[0] = subcadena;
        return i;
    }else if (operacion[0]==="muestre"){
        //muestre
        if (operacion[1]==="acumulador\r"){
            document.getElementById("screen").insertAdjacentHTML("beforeend",memoria[0]);
            //salto de liena
            document.getElementById("screen").innerHTML += "<br>";
        }else{
            document.getElementById("screen").insertAdjacentHTML('beforeend', memoria[posicion]);
            //salto de liena
            document.getElementById("screen").innerHTML += "<br>";
        }
        return i;
    }else if (operacion[0]==="imprima"){
        //imprima
        if(operacion[1]==="acumulador\r"){
            document.getElementById("print").insertAdjacentHTML("beforeend",memoria[0]);
            //salto de liena
            document.getElementById("print").innerHTML += "<br>";
        }else{
            document.getElementById("print").insertAdjacentHTML('beforeend', memoria[posicion]);
            //salto de liena
            document.getElementById("print").innerHTML += "<br>";
        }
        return i;
    }else if(operacion[0]==="vaya"){
        //vaya
        let posicion = diccionarioProcesos[idDocumento][1][variable];
        i=posicion;
        return i;
    }else if(operacion[0]==="vayasi"){
        //vayasi
        if(memoria[0] < 0){
            let posicion = diccionarioProcesos[idDocumento][1][quitarSlash(operacion[2])];
            console.log(posicion);
            i=posicion-1;
            return i;
        }else if(memoria[0] > 0){
            let posicion = diccionarioProcesos[idDocumento][1][operacion[1]];
            i=posicion-1;
            return i;
        }
        return i;
    }else if(operacion[0]==="Y"){
        //Y
        let variable1 = diccionarioProcesos[idDocumento][0][quitarSlash(operacion[1])];
        let variable2 = diccionarioProcesos[idDocumento][0][quitarSlash(operacion[2])];
        let result = diccionarioProcesos[idDocumento][0][quitarSlash(operacion[3])];
        if(memoria[variable1] && memoria[variable2]){
            memoria[result] = 1;
        }else{
            memoria[result] = 0;
        }
        return i;
```

```
}else if(operacion[0]==="O"){
        //O
        let variable1 = diccionarioProcesos[idDocumento][0][quitarSlash(operacion[1])];
        let variable2 = diccionarioProcesos[idDocumento][0][quitarSlash(operacion[2])];
        let result = diccionarioProcesos[idDocumento][0][quitarSlash(operacion[3])];
        if(memoria[variable1] || memoria[variable2]){
            memoria[result] = 1;
        }else{
            memoria[result] = 0;
        }
        return i;
    }else if(operacion[0]==="NO"){
        //NO
        let variable1 = diccionarioProcesos[idDocumento][0][quitarSlash(operacion[1])];
        let result = diccionarioProcesos[idDocumento][0][quitarSlash(operacion[2])];
        if(memoria[variable1]){
            memoria[result] = 0;
        }else{
            memoria[result] = 1;
        }
        return i;
    }else if(operacion[0]==="reinicio"){
        //reinicio de variable
        if(isNaN(memoria[posicion])){
            memoria[posicion]="";
        }else{
            memoria[posicion]=0;
        }
        return i;
    }else{
        return i;
    }
}
```

Ventana de continuar

Presenta una ventana en pantalla la cuál permite la elección si continuar en modo paso a paso o no. Contiene un condicional para realizar el cambio de la variable “pasoapaso”.

```
// ventana para confirmar si se desea continurar en modo paso a paso
function ventanaContinuar(){
    if(confirm("¿Desea seguir en modo paso a paso?")){
        pasoapaso=true;
    }else{
        pasoapaso=false;
    }
}
```

Ventana de pausa

Enseña una alerta la cual indica que la ejecución de los procesos se ha pausado.

```
// ventana para informar que se pausó la ejecución
function ventanaPausar(){
    alert("Se ha pausado la ejecución");
}
```

Iniciar ejecución de los procesos

Función que contiene un ciclo que va hasta el tamaño asignado a la memora principal, llamando las funciones respectivas para la correcta ejecución de los procesos. Si el tamaño del vector “proceos” es mayor a 0, entra a realizar el llamado de los procesos y los cambios en las variables requeridos, si no, realiza un cambio a la variable indicadora del ciclo para finalizarlo. El ciclo contiene un setTimeout para realizar una espera de 500 milisegundo antes de continuar.

```
// función para inicializar la ejecución de los procesos
async function iniciar() {
    //cambiar modo del pograma
    document.getElementById("programMode").innerHTML = "Usuario";
    for (let i = 0; i < (memoria.length); i++) {
        if (indicador<=contador && indicador<memoria.length){
            //ejecutar proceso
            console.log(indicador,memoria[indicador]);
            indicador=ejecutar(indicador);
            mostrarMemoria();
            indicador++;
        }else{
            //habilitar boton ejecutar
            botonEjecutarAuto.removeAttribute("disabled");
            i=memoria.length;
        }
        if (pasoapaso==true){
            ventanaContinuar();
        }
        await new Promise(resolve => setTimeout(resolve, 700));
    }
}
```

Selección algoritmos de planificación de procesos

Consulta el valor de planificación de procesos que esté seleccionado en la interfaz gráfica y lo retorna

```
function seleccionarProceso(){
    let select = document.getElementById("procesoElegido");
    let valor = select.value;
    return valor;
}
```

Indicador de instrucción a ejecutar

Función que retorna el valor de la primera posición del primer vector (“inicio”) en el vector llamado “proceso”, si el tamaño de este vector es mayor a 0.

```
function indicadorProcesos(){
    if (proceso.length>0){
        return proceso[0][0];
    }
}
```

Eliminar un proceso

Función que elimina un proceso (representado por un vector), si el valor del número almacenado en la última posición (“fin") es igual al valor del número almacenado en la primera posición (“inicio”).

```
function eliminarProceso(){
    if (proceso[0][0]===proceso[0][proceso[0].length-1]){
        secuencia=0;
        terminoProceso=true;
        proceso.shift();
    }
}
```

Cambio de posición en vector

Función que realiza el cambio de posición (“rafaga”) con la posición (“tiempoDeEjecucion”) en el vector que esté en la primera posición del vector “proceso”.

```
function Cambio(){
    let temp=proceso[0][proceso[0].length-3];
    proceso[0][proceso[0].length-3]=proceso[0][proceso[0].length-5];
    proceso[0][proceso[0].length-5]=temp;
}
```

Elección de algoritmo de planificación de procesos

Función que, según el valor seleccionado por el usuario, realiza la acción pertinente respecto a lo solicitado.

```
function algoritmosProceso(){
    let select = seleccionarProceso();
    if(select==="2"){
        roundRobin();
        quantum++;      You, la semana pasada • se imnplementan los algoritmos de manejo de proce…
    }else if (select==="3"){
        if (terminoProceso===true){
            proceso.sort(compararRafaga);
        }
    }else if(select==="4"){
        compararRafagaTiempo(proceso, tiempoRestante);
    }else if(select==="5"){
        if(terminoProceso===true){
            proceso.sort(compararPrioridad);
            envejecimiento();
        }
    }else if(select==="6"){
        proceso.sort(compararPrioridad);
        envejecimiento();
    }
}
```

Comparar ráfaga

Compara el valor de la posición (“ráfaga”) de los demás vectores (procesos) con el valor de la posición (“tiempoDeEjecución”) del proceso en ejecución. Si encuentra un valor menor, realiza el cambio y pone el vector (procesos) en la primera posición del vector “proceso” y lo retorna actualizado.

```
function compararRafagaTiempo(proceso, tiempo) {      You, la semana pasada • se imnplementan los algoritmos de manejo de proce…
    // Encuentra el índice del vector con la posición 5 de menor valor que tiempo
    let menorIndice = 0;
    for (let i = 1; i < proceso.length; i++) {
      if (proceso[i][proceso[i].length-5] < proceso[menorIndice][proceso[i].length-5] && proceso[i][proceso[i].length-5] < tiempo) {
        menorIndice = i;
      }
    }

    // Intercambia el vector en la posición 0 con el vector en el índice encontrado
    if (menorIndice !== 0) {
      const temp = proceso[0];
      proceso[0] = proceso[menorIndice];
      proceso[menorIndice] = temp;
    }

    // Retorna el vector de vectores actualizado
    return proceso;
}
```

Comparar prioridad

Función que comprara el valor “prioridad” de los vectores (procesos) y los ordena de manera ascendente.

```
function compararPrioridad(a,b){
    return b[a.length-2] - a[b.length-2];
}
```

Envejecimiento

Función que aumenta la prioridad de los procesos para evitar la inanición.

```
function envejecimiento(){
    for (let i = 1; i < proceso.length; i++) {
        if(proceso[i][proceso[i].length-2]<99){
            proceso[i][proceso[i].length-2]++;
        }
    }
}
```

RoundRobin

Función que compara si es igual el valor Quantum elegido por el usuario con la variable “quantum” para realizar un desplazamiento a la izquierda de los vectores (procesos) e inicia la variable “quantum” en 0.

```
function roundRobin(){
    //traer el valor del quantum
    if (quantum===parseInt(sliderQ.value)){
        desplazarIzquierda(proceso);
        quantum=0;
    }
}
```

Desplazar hacia la izquierda

Función que recibe el vector (proceso), elimina el primer vector(procesos) dentro del vector (proceso) y lo almacena en la última posición

```
function desplazarIzquierda(proceso) {
    const primero = proceso.shift();      You, la semana pasada • se imnplementan los algoritmos de manejo de pr
    proceso.push(primero);
}
```

Iniciar ejecución de los procesos

Función que contiene un ciclo que va hasta el tamaño asignado a la memora principal, llamando las funciones respectivas para la correcta ejecución de los procesos. Si el tamaño del vector “proceos” es mayor a 0, entra a realizar el llamado de los procesos y los cambios en las variables requeridos, si no, realiza un cambio a la variable indicadora del ciclo para finalizarlo. El ciclo contiene un setTimeout para realizar una espera de 500 milisegundo antes de continuar.

```
async function iniciar() {
    document.getElementById("programMode").innerHTML = "Usuario";  //cambiar modo de ejecución a usuario
    sliderQ.disabled=true;  //deshabilitar slider de quantum
    for (let i = 0; i < (memoria.length); i++) {
        if (proceso.length>0){
            tiempoRestante=proceso[0][proceso[0].length-3];
            algoritmosProceso();
            terminoProceso=false;
            memoria[0]=proceso[0][proceso[0].length-6];     //acumulador de proceso
            indicador=indicadorProcesos();    //obtener indicador de proceso de la lista de procesos
            indicador=ejecutar(indicador);   //ejecutar proceso y obtener indicador
            indicador++;   //aumentar indicador de proceso      You, hace 6 días • se agrega función para mostrar lista de proce
            proceso[0][0]=indicador;   //asignar indicador a la primera posición del proceso
            eliminarProceso();
            mostrarMemoria();
            mostrarProcesosEspera();
        }else{
            botonEjecutarAuto.removeAttribute("disabled");  //habilitar boton de ejecución automatica
            sliderQ.disabled=false;  //deshabilitar slider de quantum
            document.getElementById("procesosEspera").innerHTML = "";
            i=memoria.length;

        }
        if (pasoapaso==true){
            ventanaContinuar();     //ventana para confirmar si se desea continuar en modo paso a paso
        }
        // document.getElementById("procesosEspera").innerHTML = ""
        await new Promise(resolve => setTimeout(resolve, 500));
    }
```

Mostrar los procesos en lista de espera

Esta función muestra en pantalla el nombre los procesos que se encuentran después de la posición 0 del vector “proceso”

```
function mostrarProcesosEspera(){
    let procesosEspera="";
    if (proceso.length>1){
        for (let i = 1; i < proceso.length; i++) {
            procesosEspera=procesosEspera+"<br>"+proceso[i][1];
        }
    }
    document.getElementById("procesosEspera").innerHTML = "<p>"+procesosEspera+"</p>";
}
```