Universidad Autónoma de Aguascalientes

Ingeniería en Sistemas Computacionales

Departamento De Sistemas Electrónicos

Sistemas Operativos

"Reporte de Proyecto"

Alumnos:	**ID:**
Frida Ximena Escamilla Ramírez	321538
Rubén Eduardo Dávila Flores	261740
César Reyes Torres	261177
Estefeen Sandoval Rodríguez	261527

Profesor: Javier Santiago Cortés López

INDICE

Introducción

En este proyecto, el usuario podrá ver simulada la funcionalidad de las técnicas de colocación de memoria y planificación de procesos, como lo son Round Robin y Buddy System. De esta manera, el usuario puede familiarizarse con estos conceptos y entender de qué manera trabajan, sus características, acompañado de una interfaz que pueda ayudar al entendimiento de este mismo.

Implementación Buddy System

Conceptual

Es un algoritmo de administración de memoria que une los agujeros adyacentes muy rápidamente cuando un proceso se aborta o se envía al disco. Simplemente divide la memoria en fragmentos de 2 KB y tiene la ventaja de conocer el posible tamaño de los fragmentos cargados, pero la desventaja es que todas las solicitudes deben redondearse a 2 sin desbordamiento. Entonces cuando un proceso se carga a la memoria, busca el primer espacio libre y se encuentran los siguientes casos:

-Si el bloque libre es del tamaño necesario del proceso, se carga.
-Si el bloque libre es de menor tamaño al necesario, se intenta buscar un próximo bloque libre.
-Si este bloque es de tamaño superior al necesario, se divide en dos partes iguales y se vuelven a realizar todas las verificaciones de casos anteriormente mencionados.

Cuando el proceso finaliza su ejecución o se descarga de memoria, las posiciones que estaban contiguas se vuelven a unir formando un bloque con un tamaño mayor.

Figura 1. Ejemplo de Buddy System

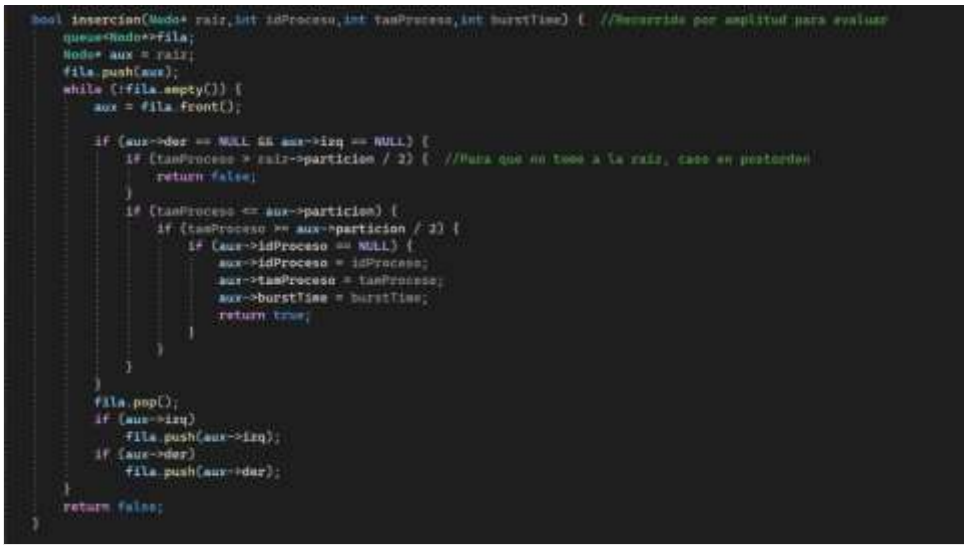
Código

```
bool asignacion(int idProceso,int tamProceso,int burstTime) {
    int n=0;
    //Calcular numero max de veces en donde pueden haber particiones (Necesario)
    if (raiz->particion > 2048) {  //<---------------------- Si el tamaño de memoria es 4096
        n = 128;
    }
    else {
        if (raiz->particion > 1024) {  //<---------------------- Si el tamaño de memoria es 2048
            n = 64;
        }
        else {  //<---------------------------------------- Si el tamaño de memoria es 1024
            n = 32;
        }
    }
    int cuanto = 27;  //El tiempo que da el CPU
    int bandera = false;
    for (int i = 0; i < n; i++) {  //Ciclo hasta que se inserte, o no
        if (insercion(getRaiz(), idProceso, tamProceso, burstTime) == true) {
            bandera = true;
            break;
        }
        else {
            particionar(tamProceso);
        }
    }
    //Para eliminar problema de que particiona demás
    //Funcion para que cheque si hay particiones que se puedan juntar, y que las junte
    for (int i = 0; i < n; i++) {
        concatenar();
    }
    //Funcion para sumar restar cuanto a los procesos
    restaCuanto();
    //Funcion para checar si algun proceso ya puede ser descargado
    eliminar();
    if (bandera == true) {
        //cout << endl << "Correcto" << endl;
        return true;
    }
    else {
        //cout << endl << "No Correcto" << endl;
        return false;
    }
}
```

La función “asignación” es nuestra función principal que es llamada desde el main. Esta función contiene las tres partes básicas para que funcione el buddy sytem, la primera es la de “inserción”, la cual nos regresará un valor booleano que nos servirá para saber si se logró insertar en la memoria o no, mientras no se pueda particionar, en un ciclo de tamaño del cuanto del sistema, se le estará restando al cuanto del proceso, y a la vez, se estará llamando a la función “particionar” para que en caso de ser posible, se hagan particiones más adecuadas para el proceso y así evitar gran desperdicio de memoria. La tercera función que llama es a la de “concatenar”, que nos ayuda a juntar particiones de memoria que estén libres.

La función de “restaCuanto” resta de uno en uno el burst time a todos los procesos en la memoria. Y esto sucede cada vez que se inserte un proceso en la memoria o que se le acabe el cuanto a un proceso.

La función llamada “eliminar” busca en la memoria algún proceso que ya haya acabado su tiempo de ejecución y lo saca de la memoria, este se apoya de la función concatenar por si al eliminar un proceso quedan particiones que se puedan juntar, y las junte.



La función “inserción” recorre nuestra memoria (árbol binario) buscando alguna partición desocupada en la que se pueda insertar. La condición para que este se pueda insertar es que sea una partición (nodo terminal) y que no tenga un ID almacenado. En caso de poderse o no insertar el proceso una vez recorrida la memoria, la función regresa un valor booleano que servirá para la función que la haya llamado.

La función “particionar” como su nombre lo indica, particiona la memoria. Esta función recorre la memoria buscando particiones disponibles (nodos terminales disponibles), una vez encontrando un lugar lo particiona y evalúa que, si se pueda insertar el proceso, si no se puede, continua intentándolo hasta que encuentre un espacio adecuado y lo inserta, en caso contrario solo se termina la función.

La función “concatenar” nos ayuda a reestablecer bloques de memoria que hayan sido desocupados y así juntarlos para hacerlos un solo bloque. La condición para que esto suceda, es que las particiones (en este caso los nodos terminales de un árbol) no tengan asignado un ID, lo que indicará que no existe algún proceso en esa partición y será necesario juntarlas (eliminando dos nodos terminales con un padre en común, convirtiéndose así, su padre, en un nuevo nodo terminal).

Implementación de Round Robin

Conceptual

El algoritmo de Round Robin es un método para asignar una tarea a la CPU. En la programación por turnos, cada tarea lista se ejecuta paso a paso solo en una cola cíclica durante un intervalo de tiempo limitado. Este algoritmo también ofrece una ejecución de procesos sin inanición.

Dentro de las características de este algoritmo es que se tienen procesos en los cuales cada uno tiene un quantum(q) de tiempo, una cola de procesos gestionada a partir de FIFO, un numero N de procesos y un tiempo de respuesta máximo que corresponde a q(N-1).

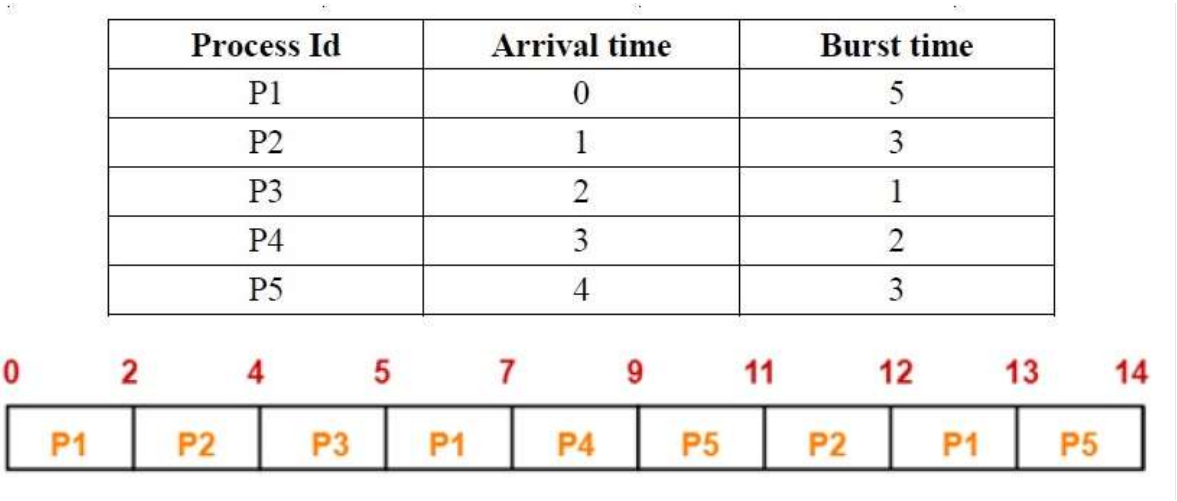
Y es a partir de Round Robin que se incluyen 7 conceptos principales los cuales son:

-Quantum: Este es el número máximo de ranuras para las que un proceso puede usar la CPU.
-Tiempo de llegada: Es el intervalo en el que comenzará a ejecutarse un proceso.
-Tiempo de ejecución: Como su nombre lo dice, es el tiempo que tarda el proceso en ejecutarse.
-Tiempo de finalización: Intervalo de tiempo en el que terminó el proceso.
-Tiempo de retorno: Es la suma de intervalos de tiempo que pasa desde que el proceso se lanza hasta que finaliza su ejecución. Se podría de es la suma del tiempo de espera más el tiempo de ejecución.
-Tiempo de espera: Es el tiempo el cual un proceso pasa dentro de la cola de listos antes de ejecutarse.

-Tiempo de respuesta: Tiempo que pasa desde que se manda a ejecuta el proceso hasta que se ejecuta por primera vez. Tiempo que pasa desde que se manda a ejecuta el proceso hasta que se ejecuta por primera vez.

Dentro del algoritmo de Round Robin en cuanto a los procesos se tiene que si un proceso agota su quantum antes de terminar su ejecución este pasa a una cola de espera para que espere nuevamente su turno. Así mismo, si el proceso se acaba o se bloquea antes de que termine su quantum se elige un nuevo proceso.

La regla es simple, si un proceso no termina con un cuanto, ingresa a la cola de procesos listos, pero si otro proceso comienza por primera vez en el intervalo de finalización del cuanto, el proceso iniciado ingresa primero a la cola de procesos listos. Entonces si el proceso pendiente entra en cola.



Process Id	Arrival time	Burst time
P1	0	5
P2	1	3
P3	2	1
P4	3	2
P5	4	3

Figura 2. Ejemplo del funcionamiento de Round Robin

Código

```
void encolar(int memTotal,int quantum) {
    srand(time(NULL));
    Arbol* memoria = new Arbol;
    Arbol::Nodo* raiz;
    memoria->setRaiz(memTotal);
    cout << endl << endl;

    int burstTime = 0, idProceso = 0, tamProceso = 0, cuanto = 1;
    float tEj = 0, porcentajeMemUsada = 0.0, totalProcesos = 0;
    queue<proceso>cola;  //Cola de Procesos tipo struct
    proceso pp;  //Auximiar strcut para almacenar en la cola
    int id = 0, tamP = 0, bT = 0;

    float promAtencion = 0;
    float promedioTamProcesos = 0;
    float promTempEjecucion = 0;
    int velocidad = 1000;
    memoria->setVelocidad(velocidad);
    char key;
    int idaux = 0;
    while(1){
        if (_kbhit()) {
            key = _getch();
            if ((int)key == 27) {  //<------------------------ Esc para salir (mostrar estadisticas)
                system("cls");
                estadisticas(promTempEjecucion, totalProcesos, porcentajeMemUsada, promedioTamProcesos, promAtencion);
                break;
            }
            if ((int)key == 43) {//<------------------------ Configurar para aumentar velocidad con '+'
                velocidad -= 200;

                if (velocidad <= 0) {  //<------------------- Configurar para tener la mayor velocidad, pensar si dejarlo en 0 o más
                    velocidad = 50;
                    memoria->setVelocidad(velocidad);
                }
                else {
                    memoria->setVelocidad(velocidad);
                }
            }
            if ((int)key == 45) {//<------------------------ Configurar para disminuir velocidad con '-'
                velocidad += 200;
                memoria->setVelocidad(velocidad);
            }
            if ((int)key == 80 || (int)key == 112) {//<------------------- Letra 'p' o 'P' para detener la ejecucion
                estadisticasDurante(promTempEjecucion, promAtencion);
                SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE), 14);
                gotoxy(80, 11);
                system("pause");
                SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE), 8);
            }
        }
        system("cls");
        menu();
        idaux++;
        pp.idProceso = idaux;
        pp.tamProceso = 50 + rand() % 200; //<------------------------------ Rango entre 50 - 200
        pp.burstTime = 1 + rand() % 10;  //<------------------------------- Rango entre 1 - 10

        pp.tempEjecucion = pp.burstTime;
        cola.push(pp);

        //Auximiliares para la asignacion en la memoria
        pp = cola.front();
        id = pp.idProceso;
        tamP = pp.tamProceso;
        bT = pp.burstTime;
        tEj = pp.tempEjecucion;

        //*************************
                //MOSTRAR ESTADISTICAS TODO EL TIEMPO - Las necesarias
        totalProcesos = memoria->estadisticas(3);
        porcentajeMemUsada = memoria->estadisticas(1);
        estadisticasDurante(totalProcesos, porcentajeMemUsada, promedioTamProcesos);
        tablaProcesosCola(cola, cuanto, quantum);  //<------------------------------------ Muestra la tabla de procesos durante la ejecucion
        cout << endl << endl << "\n     Memoria Total: " << memTotal << "    Cuanto designado: " << quantum << endl;
        cout << endl << endl << "    -Representacion de la memoria-" << endl << endl;
        memoria->imprimirMemoria();
        cout << endl << endl << endl << "    -Representacion de la memoria con un arbol binario-" << endl << endl;
        memoria->printBT(memoria->getRaiz());
        //*************************

        if (memoria->asignacion(id, tamP, bT) == true) {
            if (id > 1) {
                promTempEjecucion = (promTempEjecucion + tEj) / 2;
                promedioTamProcesos = (promedioTamProcesos + pp.tamProceso) / 2;
                promAtencion = (promAtencion + cuanto) / 2;
            }
            else { //Cuando es el primer proceso, el promedio es el valor inicial
                promTempEjecucion = tEj;
                promedioTamProcesos = pp.tamProceso;
                promAtencion = cuanto;
            }
            cola.pop();
            //Cuanto general pero que funciona de manera particular para cada proceso
            cuanto = 0;  //<---- Se reinicia en caso de que el proceso ya se haya insertado en la memoria
        }
        else {  //Sigue aumentando solo si el proceso aun no puede ingresar a la memoria (El cuanto aumenta hasta el limite)
            cuanto++;
            pp.tempEjecucion++;
        }
        if (cuanto == quantum) {  //<------------- Si ya pasó el cuanto asignado, volver a formar el proceso en la cola
            cola.push(cola.front());
            cola.pop();
            cuanto = 0;  //Reinicia el cuanto
        }

        Sleep(velocidad);
    }
}
```

Dentro de la función llamada “encolar” se encuentra la técnica de round robin, además de otras cosas necesarias para su funcionamiento, en este caso prestemos atención en la siguiente parte.

```
idaux++;
pp.idProceso = idaux;
pp.tamProceso = 50 + rand() % 200; //<---------------------------- Rango entre 50 - 200
pp.burstTime = 1 + rand() % 10;  //<---------------------------- Rango entre 1 - 10

pp.tempEjecucion = pp.burstTime;
cola.push(pp);

//Auximiliares para la asignacion en la memoria
pp = cola.front();
id = pp.idProceso;
tamP = pp.tamProceso;
bT = pp.burstTime;
tEj = pp.tempEjecucion;
```

Aquí es donde se crean los procesos de manera aleatoria, con un tamaño aleatorio, un id asignado y un burst time aleatorio. Después de crearse, los procesos se encolan en una estructura de datos llamada cola, que literalmente funciona de esa manera. Después se utilizan auxiliares para poder insertar estos procesos en la memoria.

```
if (memoria->asignacion(id, tamP, bT) == true) {
    if (id > 1) {
        promTempEjecucion = (promTempEjecucion + tEj) / 2;
        promedioTamProcesos = (promedioTamProcesos + pp.tamProceso) / 2;
        promAtencion = (promAtencion + cuanto) / 2;
    }
    else { //Cuando es el primer proceso, el promedio es el valor inicial
        promTempEjecucion = tEj;
        promedioTamProcesos = pp.tamProceso;
        promAtencion = cuanto;
    }
    cola.pop();
    //Cuanto general pero que funciona de manera particular para cada proceso
    cuanto = 0;  //<---- Se reinicia en caso de que el proceso ya se haya insertado en la memoria
}
else {  //Sigue aumentando solo si el proceso aun no puede ingresar a la memoria (El cuanto aumenta hasta el limite)
    cuanto++;
    pp.tempEjecucion++;
}
if (cuanto == quantum) {  //<---------- Si ya pasó el cuanto asignado, volver a formar el proceso en la cola
    cola.push(cola.front());
    cola.pop();
    cuanto = 0;  //Reinicia el cuanto
}

Sleep(velocidad);
}
}
```

En esta sección que es la última parte de la función, es donde se toma un proceso de la cola y se intenta insertar en la cola, tomando en cuenta el cuanto que se ha asignado, si se logra insertar antes de que su cuanto termine, nos indicará que se logró insertar y el proceso será eliminado de la cola. En caso contrario, si acaba su cuanto antes de que se pueda insertar, se manda el proceso al final de la cola y se continua con el siguiente en la cola.

Conclusiones

César Reyes Torres

Con ese proyecto se concluye el curso de sistemas operativos. Una vez terminado este proyecto puedo decir que se lograron los objetivos, tanto del proyecto como de la materia en general. Es muy interesante como es que funcionan los diferentes sistemas operativos y de que se conforman. Hace un tiempo no tenía ni idea de estas técnicas de planificación y de acomodo en la memoria, pero, con este proyecto, pude reforzar el aprendizaje en clase sobre round robin y buddy system, algo que sonaba tan complejo, pero aplicando los conocimientos de otras materias se vuelve muy fácil construirlo. Este proyecto represento todo un reto, pero al final de cuentas se logró terminar con todas las especificaciones requeridas, estoy muy satisfecho.

Frida Ximena Escamilla Ramírez

Gracias a este proyecto, he podido reforzar los conocimientos que previamente conocíamos por la clase, sin embargo, ha sido muy importante poder implementar estos conceptos en un código, que realizara lo que en teoría sabíamos, y que pudiera simular el funcionamiento para ayudarnos a comprender todo de una mejor manera.

Este curso fue de mucho aprendizaje para mí, ya que no sabía muchas de las cosas que se consideraban básicas, y más que eso, tuve la oportunidad de aprender cosas nuevas, y reforzar las que ya sabía, de manera que me gustó el curso en general y las actividades presentadas en él.

Estefeen Sandoval Rodriguez

En el transcurso de este proyecto creo que se han reforzado todos los conocimientos acerca de cómo funcionan los sistemas operativos y sus diferentes módulos, como lo son el planificador de procesos que en esta ocasión implementamos el famoso Round Robin, y de administrador de memoria como lo es el Buddy System, así que siento que fue un excelente cierre para este curso tan completo lleno de detalles e información del funcionamiento básico de algo tan complejo como lo es un sistema operativo.

Rubén Eduardo Dávila Flores

Después de la finalización de este proyecto, personalmente puedo decir que fue muy importante la realización de un proyecto de este tipo ya que principalmente sirvió para implementar de una forma real algunos de los conceptos que durante el transcurso de las clases estuvimos viendo de una forma mucho más teórica, ya que en este caso se nos fue posible hacer el intento de como estar simulando una memoria a través del uso de algoritmos que en este caso los indicados fueron los de Round Robin para la parte de la creación de una cola de procesos y por otro lado

Buddy System en donde se lograba meter estos procesos dentro de una memoria que se iba particionando de acuerdo a las condiciones necesarias.

Conclusión General del Equipo

En conclusión, el estudio de los sistemas operativos en este curso nos ha brindado una base valiosa para comprender los sistemas complejos que alimentan nuestras computadoras y otros dispositivos que ya conocemos. A través de nuestra exploración de temas como la gestión de procesos, la gestión de memoria, entre otros, hemos obtenido una apreciación más profunda del intrincado funcionamiento interno de la tecnología moderna. También hemos aprendido cómo diseñar e implementar sistemas operativos eficientes y efectivos, y cómo detectar y resolver problemas comunes que puedan surgir en el camino. En general, este curso ha sido una valiosa experiencia de aprendizaje para todos nosotros, y que nos ha equipado con el conocimiento y las habilidades necesarias para tener éxito en el campo de la informática, los sistemas operativos y la programación.
En lo que concierne al proyecto, a través de nuestro análisis e implementación del buddy system y el round robin, hemos obtenido una comprensión más profunda de cómo funcionan estos algoritmos y cómo se pueden aplicar en la práctica. También hemos aprendido sobre sus ventajas y desventajas, y cómo se comparan con otros algoritmos de asignación de memoria.

Bibliografía

Bhatele, S. (2022, 30 abril). *Introduction to Round Robin Scheduling Algorithm (C++ and Java Code)*. Medium. Recuperado 13 de diciembre de 2022, de https://levelup.gitconnected.com/introduction-to-round-robin-scheduling-algorithm-c-and-java-code-fa1909dd46ab

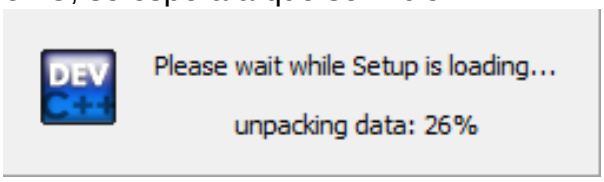
Explicación De Round Robin Planificación De Procesos. (2020, 9 mayo). Codwelt. Recuperado 13 de diciembre de 2022, de https://blog.codwelt.com/explicacion-de-round-robin-planificacion-de-procesos/

Buddy System. (s. f.). Universidad Autónoma de Yucatán. Recuperado 13 de diciembre de 2022, de https://www.coursehero.com/file/p4u4n3u/Buddy-System-Es-un-algoritmo-de-manejo-de-memoria-que-hace-que-sea-muy-r%C3%A1pida/

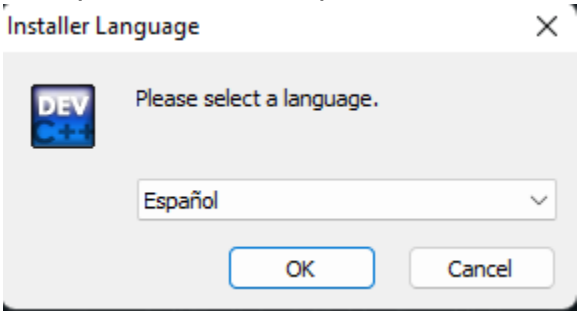
Anexo A

-Primeramente, dentro de la carpeta se selecciona para ejecutar la aplicación Dev C++.exe

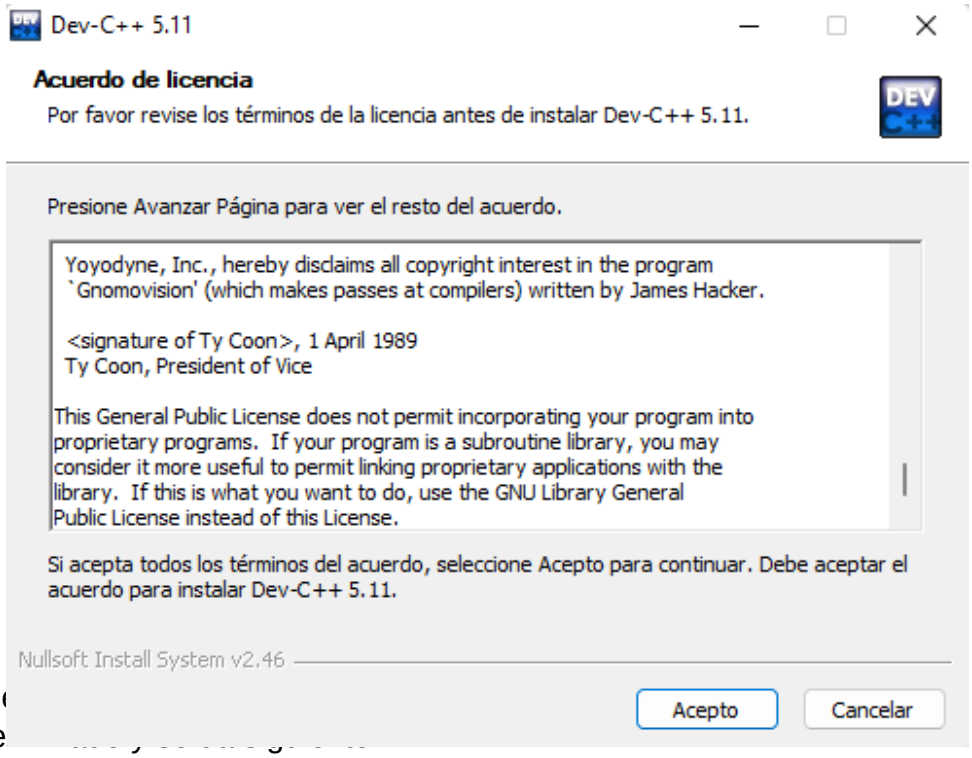
-Configuración del Dev C, se espera a que se inicie



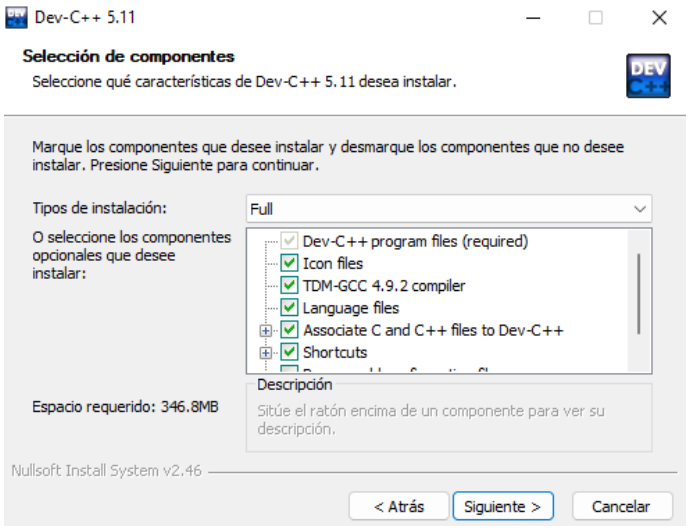
-Se selecciona el idioma, que en este caso para su uso fácil se deja como español



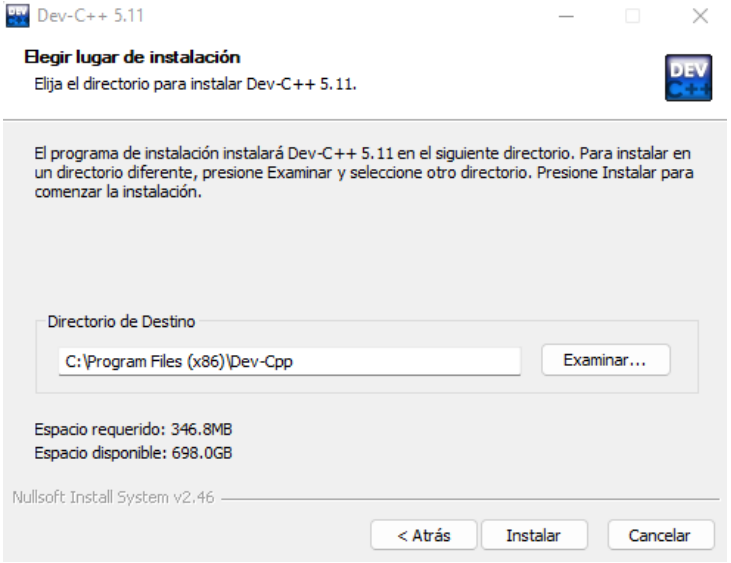
-Se aceptan los acuerdos de licencia



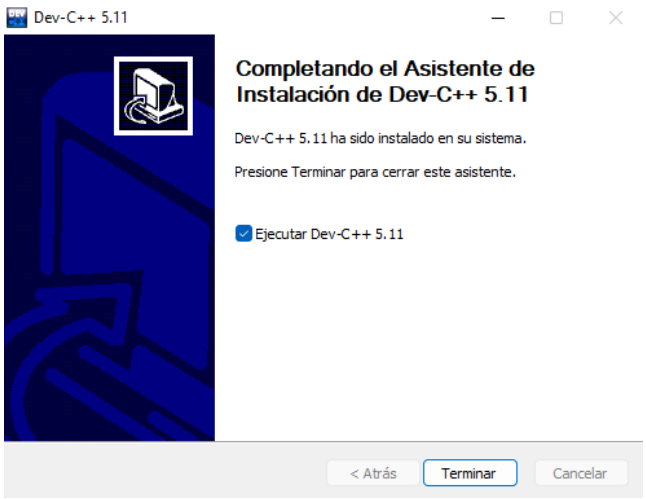
-Despu
predete



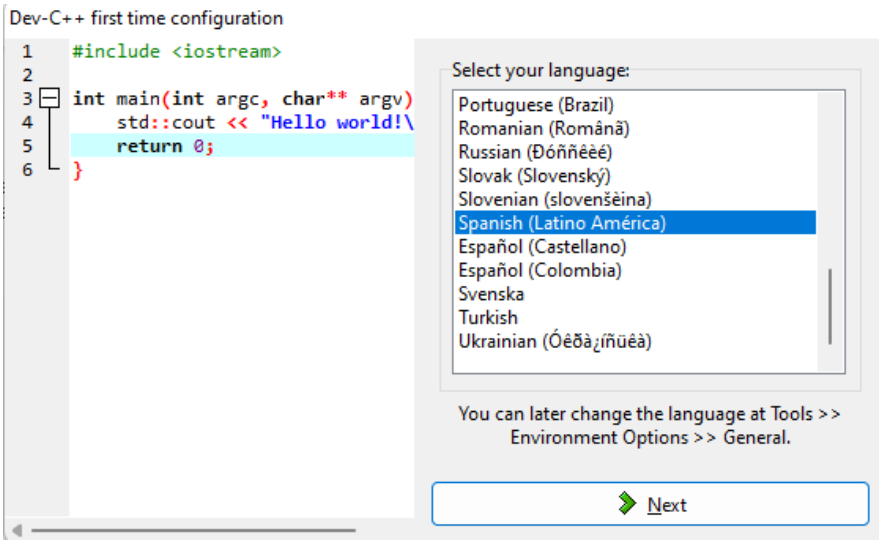
-Se selecciona el lugar de instalación dentro del equipo, en este caso usted puede seleccionar el lugar de su preferencia.



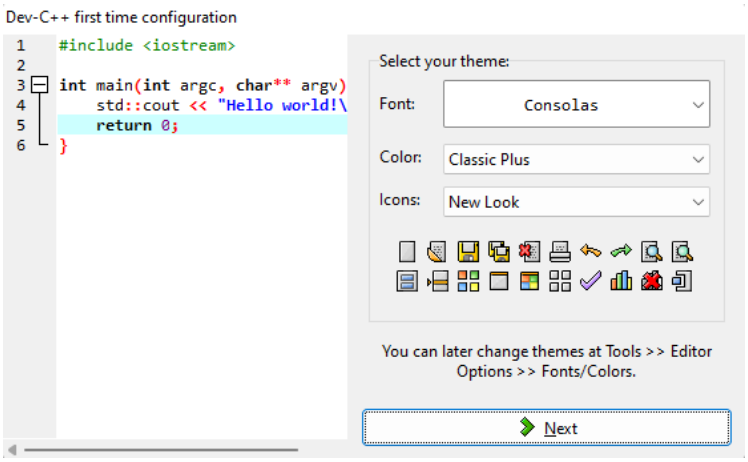
-Se espera a que se instalen todos los paquetes necesarios y de da por finalizada la instalación, dejando seleccionada la opción de Ejecutar Dev-C++.



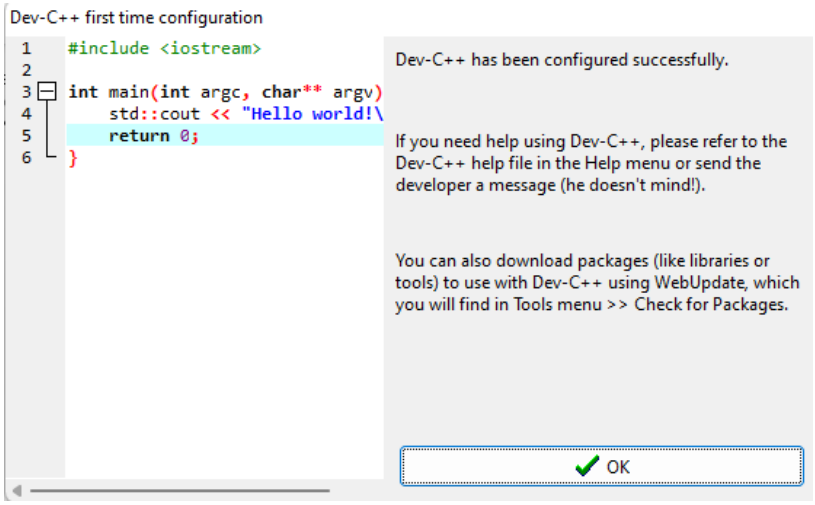
-Dentro de la misma configuración del Dev C++, se selecciona el lenguaje de la aplicación en donde se selecciona la opción de Spanish (Latino America) y se da Next.



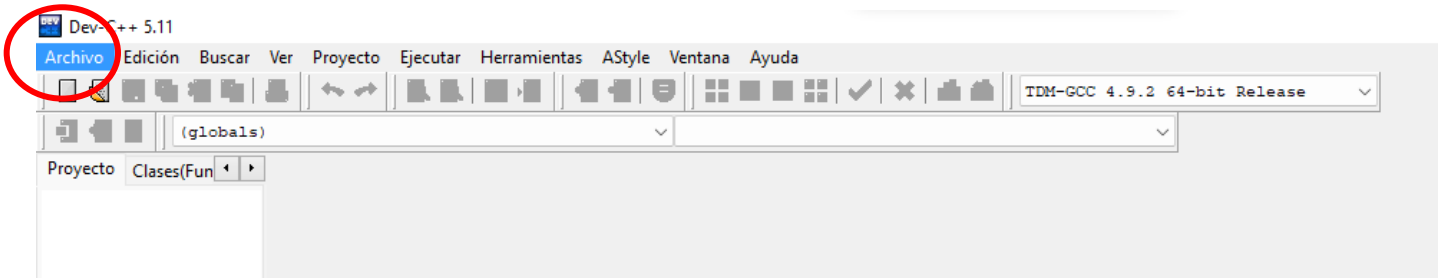
-Se selecciona el tema base de la fuente de letra, el color y los iconos. En este caso no se moverá nada y se deja todo como está predeterminado para despues dar Next.



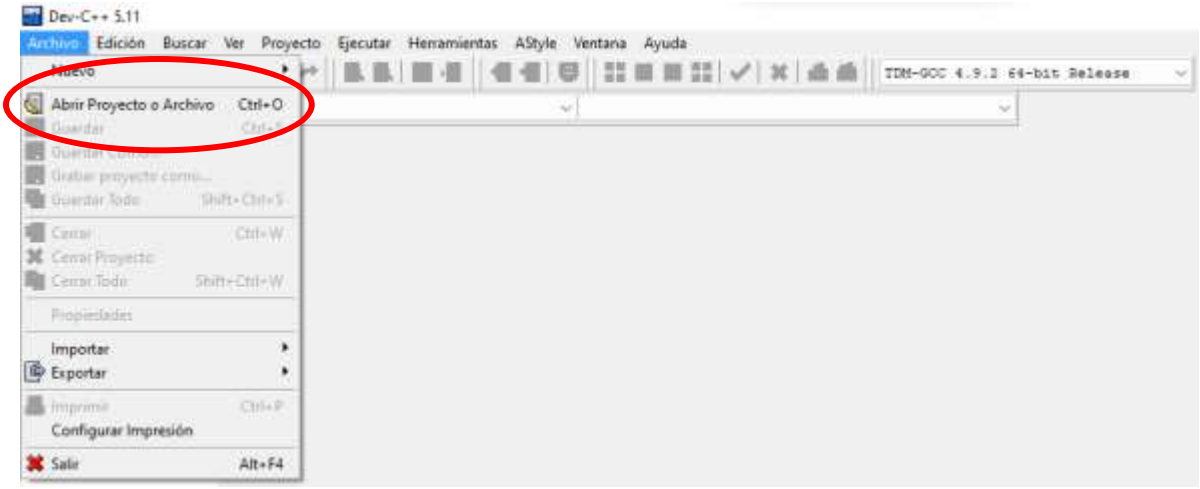
-Finalmente se termina la configuración y se da Ok



-Dentro de la aplicación Dev-C++ se va a seleccionar dentro del menú la parte de archivos

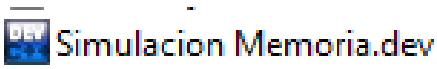


-Ahora dentro del menú de archivos seleccionaras la parte de Abrir Proyecto o Archivo

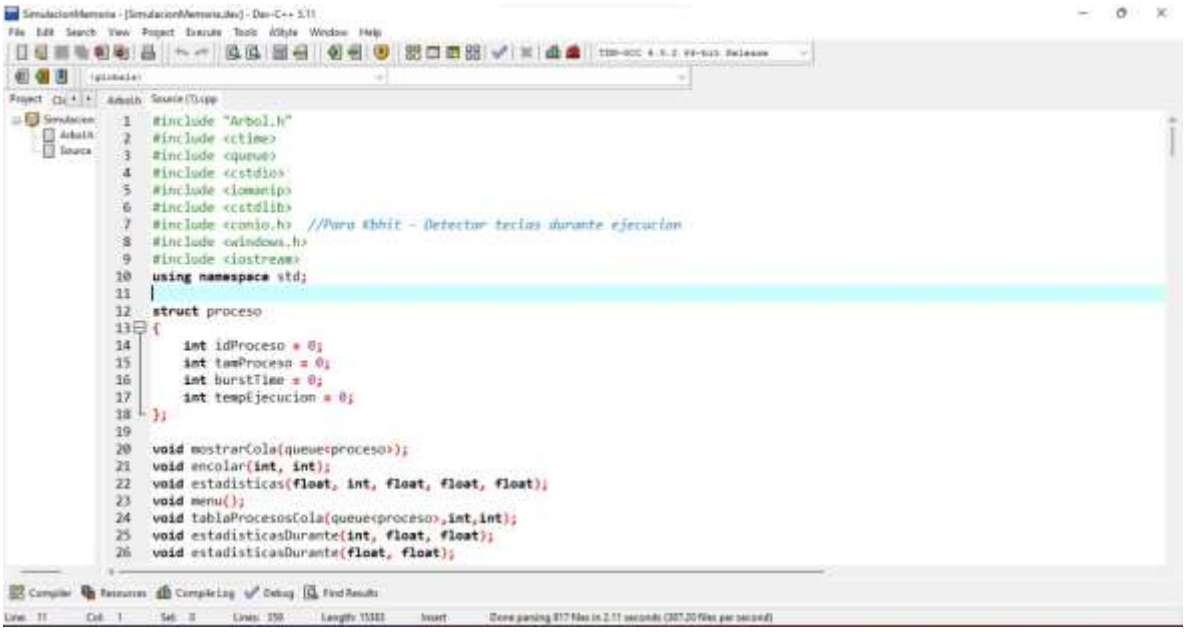


-Al abrir el Explorador de Archivos, dentro de la carpeta del proyecto de nuestro equipo, se encontrará otra carpeta llamada Código Completo a la cual accederá

-Dentro de la carpeta de Código Completo encontrará un archivo llamado Simulación Memoria.dev el cual seleccionará para abrir.



-Después dentro del Dev-C++ se abrirá nuestro proyecto con los dos archivos



```
#include "Arbol.h"
#include <ctime>
#include <queue>
#include <cstdio>
#include <iomanip>
#include <cstdlib>
#include <conio.h>  //Para kbhit - Detectar teclas durante ejecucion
#include <windows.h>
#include <iostream>
using namespace std;

struct proceso
{
    int idProceso = 0;
    int tamProceso = 0;
    int burstTime = 0;
    int tempEjecucion = 0;
};

void mostrarCola(queue<proceso>);
void encolar(int, int);
void estadisticas(float, int, float, float, float);
void menu();
void tablaProcesosCola(queue<proceso>,int,int);
void estadisticasDurante(int, float, float);
void estadisticasDurante(float, float);
```

-Teniendo ya abierto el proyecto sigue simplemente ejecutar el código, pero para que se ejecute correctamente se debe tener que se vea en pantalla el archivo de de Source(1).cpp ya que el cpp es el main principal del código.

-Después de esto solo queda seleccionar la opción Ejecutar y Compilar para abrir nuestro archivo exe.

Link del Video

https://eduuaa-my.sharepoint.com/:f:/g/personal/al261177_edu_uaa_mx/EljJfQHqfOFLn2H3Ooz_9eQBWOTa4zftOyr0TGM6k4M9QA?e=SZB8Qp