INSTITUTO TECNOLOGICO DE IZTAPALAPA I

INGENIERIA EN SISTEMAS COMPUTACIONALES

Propuesta para el desarrollo del proyecto titulado:

VERONA

Presenta:

CUANENEMI CUANALO MARIO ALBERTO

FERMIN CRUZ ERIK

GUTIERREZ ARELLANO RAFAEL

PEREZ ARMAS FAUSTO ISAAC

No. De control:

181080030

181080007

181080022

181080037

ASESOR INTERNO:

M.C. ABIEL TOMAS PARRA HERNANDEZ

CIUDAD DE MEXICO **JUNIO/2021**

RESUMEN

Verona, un nuevo lenguaje desarrollado por Microsoft Research Cambridge que explora el concepto de propiedad concurrente. Si bien las CPU modernas implementan instrucciones atómicas y de ubicación de caché (además de las barreras ineficaces de la generación anterior),

Este control preciso solo es significativo para objetos pequeños (generalmente del tamaño de registro).

MLIR es una representación intermedia de varios niveles desarrollada dentro del proyecto Tensor Flow, pero luego migrada bajo el paraguas de LLVM.

Como parte central del proyecto LLVM, MLIR se está utilizando en mucho más que solo representaciones de ML y está ganando terreno como la representación de alto nivel de algunas interfaces de lenguaje (Fortran, descripción de hardware) y otros experimentos que utilizan Clang MLIR como su propio idioma intermedio.

El principal beneficio de usar MLIR para la reducción del lenguaje es que puede mantener la semántica del lenguaje tan alto como lo necesite, construyendo operaciones de dialecto que codifican la lógica que desea y luego creando pases que usan esas operaciones para inferir comportamientos y tipos o para transformarse en un formato más óptimo, antes de bajar a los dialectos estándar y más abajo, LLVM IR.

Esto soluciona los dos grandes problemas en las interfaces: por un lado, es mucho más fácil trabajar con operaciones IR flexibles que los nodos AST (árbol de sintaxis abstracta) y, por otro lado, solo bajamos a LLVM IR (que es nivel muy bajo) cuando nos sentimos cómodos no podemos extraer ningún comportamiento más especial de la semántica del lenguaje.

Otros dialectos existentes se suman a la lista de beneficios, ya que ya tienen una semántica rica y pases de optimización existentes que podemos usar de forma gratuita.

PALABRAS CLAVE:
CPU, caché, registro, Tensor Flow, LLVM, Clang

ABSTRACT

Verona, is a new language developed by Microsoft Research Cambridge that explores the concept of concurrent ownership. While modern CPUs implement atomic and cache location instructions (in addition to the ineffective barriers of the previous generation),

This fine control is only olí for small objects (usually record size).

MLIR is a multi-level intermediate representation developed within the Tensor Flow project, but later migrated under the LLVM umbrella.

As a core part of the LLVM project, MLIR is being used in much more than just ML representations and is gaining traction as the high-level representation of some language interfaces (Fortran, hardware description) and other experiments that use Clang MLIR as their own intermediate language.

The main benefit of using MLIR for language reduction is that you can keep the semantics of the language as high as you need it, building dialect operations that encode the logic you want, and then creating passes that use those operations to infer behaviors and types or to transform into a more optimal format, before downgrading to the standard dialects and below, LLVM IR.

This solves the two big problems in the interfaces: on the one hand, it is much easier to work with flexible IR operations than the AST nodes (abstract syntax tree) and on the other hand, we only go down to LLVM IR (which is very low level) when we feel comfortable we cannot extract any more special behavior from the semantics of the language.

Other existing dialects add to the list of benefits as they already have rich semantics and existing optimization passes that we can use for free

KEYWORDS:
CPU, cache, Join, Tensor Flow, LLVM, Clang

INDICE

INTRODUCCIÓN

De acuerdo con la naturaleza del funcionamiento de las computadoras, se dice que estas siempre ejecutan órdenes en un formato que les resulta inteligible; dichas órdenes se agrupan en programas, conocidos como software, el cual, para su estudio, a su vez, se divide en dos partes: el formato de representación interno de los programas, que constituye el lenguaje máquina o código ejecutable, y el formato de presentación externa, que es un archivo o un conjunto de archivos, que puede o no estar en un formato que puede ser visto/leído por el usuario (es decir, en un formato que respeta las reglas).

Para ejecutar lo que el usuario desea hacer en su computadora, o bien para resolver un problema específico, este precisa buscar un software que realice o ejecute con exactitud la tarea que se ha planteado o elaborar y desarrollar (escribir) un programa que la realice. El trabajo de elaboración de un programa se denomina “programación”. Pero la programación no es solo el trabajo de escritura del código, sino todo un conjunto de tareas que se deben cumplir, a fi n de que el código que se escribió resulte correcto y robusto, y cumpla con el objetivo o los objetivos para los que fue creado.

Las afirmaciones que se derivan de lo anterior son varias:

- Conocer las herramientas, los formalismos y los métodos para transformar un problema en un programa escrito en un lenguaje (que posiblemente no será el lenguaje máquina), y para que dicho programa pueda ser transformado en un código ejecutable.
- Saber transformar el problema inicial en un algoritmo y luego en un programa.

La primera afirmación es genérica y se considera para varias categorías de problemas para resolver. Por su parte, la segunda es específica de un problema determinado que se tiene que resolver, para lo cual existen diversas metodologías específicas de resolución para este tipo de problemas. Para los casos de problemas muy generales, en ocasiones existen métodos conocidos que solo se adaptan a un problema en particular. El método es, por lo general, un algoritmo o una técnica de programación

MLIR es una representación intermedia de varios niveles desarrollada dentro del proyecto Tensor Flow, pero luego migrada bajo el paraguas de LLVM. Como parte central del proyecto LLVM, MLIR se está utilizando en mucho más que solo representaciones de ML y está ganando terreno como la representación de alto nivel de algunas interfaces de lenguaje (Fortran, descripción de hardware) y otros experimentos que utilizan Clang MLIR como su propio idioma intermedio.

El principal beneficio de usar MLIR para la reducción del lenguaje es que puede mantener la semántica del lenguaje tan alto como lo necesite, construyendo operaciones de dialecto que codifican la lógica que desea y luego creando pases que usan esas operaciones para inferir comportamientos y tipos o para transformarse en un formato más óptimo, antes de bajar a los dialectos estándar y más abajo, LLVM IR.

JUSTIFICACION

Microsoft ha confirmado que está creando un nuevo lenguaje de programación, bajo el nombre de Proyecto Verona, inspirado en el lenguaje de programación Rust, también de Microsoft, y pensado para utilizarlo en programación segura de infraestructuras.

Según Zdnet, este lenguaje de programación no es algo nuevo, dado que ya hay noticias de su nacimiento el año pasado (2020), cuando Matthew Parkinson, un investigador del Laboratorio Cambridge Computer ofreció algunos detalles del proyecto a dicho medio. Parkinson también anunció entonces que el lenguaje sería open source.

Eso sí, en su página de Github queda claro que Proyecto Verona no es un producto y que no tiene relación con el uso que Microsoft hace de C++, C# y Rust.

Proyecto Verona no solo se inspira en Rust, sino que además toma conceptos de Cyclone, un lenguaje considerado como una especie de derivado seguro de C. También de Pony, un lenguaje que cuenta entre los que más han contribuido a desarrollarlo con integrantes de Microsoft Research.

Los de Redmond lo consideran un «lenguaje de programación de investigación y esperan que esta consideración, así como el hacerlo open source

Según Parkinson, además, con Proyecto Verona están dando forma a un área de la programación de sistemas: la programación de infraestructuras, que tiene «unos requisitos de rendimiento y predictabilidad importantes, sin necesitas acceso en bruto a la máquina

OBJETIVO

Conocer el lenguaje Verona, así como sus diferentes funciones dentro de la programación y sus usos para la tecnología mas actualizada.

En Microsoft siguen pensando *destinar el lenguaje a la programación segura de infraestructuras*, pero ahora añaden que quieren hacerlo *a través de una mejor gestión de memoria,* de la compartimentalización y de lo que podemos denominar «sandboxing ubicuo». También han manifestado que en su desarrollo están colaborando académicos del Imperial College de Londres.

Otro de los objetivos de Proyecto Verona es *contribuir a la securización de código en lenguajes inseguros, como C y C#,* que todavía hay en mucho del código heredado de Microsoft.

OBJETIVOS ESPECÍFICOS

Objetivo Específico 1.

Encontrar y definir diferencias entre los lenguajes de programación de codigo abierto

Objetivo Específico 2.

Definir los métodos de administración de memoria

Objetivo Específico 3.

Definir las características de los lenguajes de programación RUST y CYCLONE

Objetivo Específico 4.

Encontrar las similitudes entre ambos lenguajes de programación para poder encontrar los motivos por los cuales fueron selecciones esos lenguajes

Objetivo Específico 5.

Definir los tipos de memoria dentro de una computadora

Objetivo Específico 6.

Realizar un análisis de los registros de memorias.

Objetivo Específico 7.

Proponer una solución alterna ante el lenguaje de programación.

METODOLOGIA

Para este trabajo, se decidió hacer uso de la metodología ágil, dado a que en esta hallamos un amplio espectro de trabajo para la investigación Por definición, las metodologías ágiles son aquellas que permiten adaptar la forma de trabajo a las condiciones del proyecto, consiguiendo flexibilidad e inmediatez en la respuesta para amoldar el proyecto y su desarrollo a las circunstancias específicas del entorno. Para esto, hemos considerado las características como:

- *Mejora de la calidad del producto:* Fomenta el enfoque proactivo de los miembros del equipo en la búsqueda de la excelencia del producto.
- *Mayor satisfacción del cliente:* El cliente está más satisfecho al verse involucrado y comprometido a lo largo de todo el proceso de desarrollo. Mediante varias demostraciones y entregas.
- *Mayor motivación de los trabajadores:* Los equipos de trabajo autogestionados, facilitan el desarrollo de la capacidad creativa y de innovación entre sus miembros.
- *Trabajo colaborativo:* La división del trabajo por distintos equipos y roles junto al desarrollo de reuniones frecuentes, permite una mejor organización del trabajo.
- *Mayor control y capacidad de predicción:* La oportunidad de revisar y adaptar el producto a lo largo del proceso ágil, permite a todos los miembros del proyecto ejercer un mayor control sobre su trabajo, cosa que permite mejorar la capacidad de predicción en tiempo y costes.

Reducción de costes: La gestión ágil del proyecto elimina prácticamente la posibilidad de fracaso absoluto en el proyecto, porque los errores se van identificando a lo largo del desarrollo en lugar de esperar a que el producto esté acabado y toda la inversión realizada.

- ***Extreme Programming XP***

Esta herramienta es muy útil sobre todo para startups o empresas que están en proceso de consolidación, puesto que su principal objetivo es ayudar en las relaciones entre los empleados y clientes. La clave del éxito del Extreme Programming XP es potenciar las relaciones personales, a través, del trabajo en equipo, fomentando la comunicación y eliminando los tiempos muertos.

Sus principales fases son:

- Planificación del proyecto con el cliente
- Diseño del proyecto
- Codificación, donde los programadores trabajan en pareja para obtener resultados más eficientes y de calidad
- Pruebas para comprobar que funcionan los códigos que se van implementando

MARCO TEÓRICO

Antecedentes

Verona tiene como objetivo arreglar las colas pasando la propiedad de una región mutable de la memoria como un mensaje. Las colas de mensajes están completamente libres de bloqueos (utilizando estructuras de datos sin bloqueos de Atomic-Swap) y solo pasan el punto de entrada a una región (una referencia aislada a todo el blob de memoria mutable) como propiedad única.

Lo que significa que solo el hilo que tiene ese objeto aislado puede acceder a cualquier memoria en la región contenida dentro. Por lo tanto, cada hilo tiene la fuerte garantía de que nadie más está accediendo a toda la región y no hay posibilidades de mutación simultánea ni necesidad de bloqueos.

Esto soluciona los dos grandes problemas en las interfaces: por un lado, es mucho más fácil trabajar con operaciones IR flexibles que los nodos AST (árbol de sintaxis abstracta) y, por otro lado, solo bajamos a LLVM IR (que es nivel muy bajo) cuando nos sentimos cómodos no podemos extraer ningún comportamiento más especial de la semántica del lenguaje. Otros dialectos existentes se suman a la lista de beneficios, ya que ya tienen una semántica rica y pases de optimización existentes que podemos usar de forma gratuita.

- **¿Por qué Verona necesita MLIR?**

Verona pretende ser fácil de programar, pero potente para expresar semántica de tipo rico y concurrente sin esfuerzo. Sin embargo, el sistema de tipos de Verona no es nada sencillo. Los lenguajes similares a C generalmente tienen tipos nativos (integer, float, boolean) que se pueden representar directamente en el hardware, o son fáciles de operar mediante una biblioteca en tiempo de ejecución; y tipos de contenedores (listas, conjuntos, colas, iteradores), que ofrecen diferentes vistas y acceso a tipos nativos.

Algunos lenguajes también admiten genéricos, que es una parametrización de algunos tipos sobre otros tipos, por ejemplo, un contenedor de cualquier tipo.

- **Características**

- Capacidad de tipo (mutable, inmutable, aislada).
- Controlar el acceso a los objetos con respecto a la mutabilidad
    - Por ejemplo, los objetos inmutables se almacenan en la memoria inmutable fuera de las regiones mutables) así como los centinelas de la región (aislados) que no pueden ser retenidos por más de una referencia de fuera de la región. Uniones de tipo (A | B).

Esta característica permite a los usuarios crear características que funcionan con varios tipos, lo que permite restringir fácilmente los tipos pasados y hacer coincidir tipos específicos en el código (mediante la concordancia de palabras clave) con la garantía de que el tipo será uno de esos.

- Escriba intersecciones (A y B). Esto permite restringir tipos con capacidades, dificultando el acceso inesperado, por ejemplo, devolviendo referencias inmutables o identificando objetos aislados en la creación.
- También puede ayudar a diseñar interfaces, creando requisitos sobre los objetos, Pero también como argumentos de función, controlando el acceso a los objetos recibidos.
- Tipos inferidos fuertes. El compilador emitirá un error si los tipos no se pueden identificar en el momento de la compilación, pero los usuarios no necesitan declararlos en todas partes y, a veces, ni siquiera pueden ser conocidos hasta que el compilador ejecuta su propio pase de inferencia de tipos (por ejemplo, genéricos, uniones o lambdas).

Verona actualmente usa un analizador PEG que produce el AST y es bastante sencillo, pero una vez construido, trabajar con AST (más específicamente creando nuevos nodos, reemplazando o actualizando los existentes) es bastante complicado y propenso a errores.

- **Funciones importantes en la programación**

While, ***for***, palabras claves que controlan el acceso a las regiones (solicita la propiedad), palabra clave de coincidencia, que controla los tipos (por ejemplo, de una unión de tipos) en el siguiente bloque

También necesitamos exponer alguna biblioteca de tiempo de ejecución mínima escrita en Verona para operar en tipos (por ejemplo, aritmética y lógica de enteros / coma flotante), y necesitamos esas clases compiladas y expuestas al código de usuario como un módulo MLIR, para que podamos mirar en el código como un

Conceptos

- **Compilador**

Es un Software que traduce un programa escrito en un lenguaje de programación de alto nivel (C / C ++, COBOL, etc.) en lenguaje de máquina. Un compilador generalmente genera lenguaje ensamblador primero y luego traduce el lenguaje ensamblador al lenguaje máquina. Una utilidad conocida como «enlazador» combina todos los módulos de lenguaje de máquina necesarios en un programa ejecutable que se puede ejecutar en la computadora.

¿En qué se diferencia de un Intérprete?

Un intérprete lee un programa fuente ejecutable, escrito en un lenguaje de programación de alto nivel, así como datos para este programa, y ejecuta el programa contra los datos para producir algunos resultados.

Hay que tener en cuenta que tanto los intérpretes como los compiladores (como cualquier otro programa) están escritos en un lenguaje de programación de alto nivel (que puede ser diferente del idioma que aceptan) y se traducen en código máquina. Un intérprete generalmente es más lento que un compilador porque procesa e interpreta cada enunciado de un programa tantas veces como el número de evaluaciones de esta afirmación. El intérprete de Java, llamado Java Virtual Machine (JVM), en realidad puede interpretar códigos de bytes directamente o puede compilarlos internamente en código máquina y luego ejecutar ese código.

Los compiladores son procesos complejos debido a que tienen varias fases por las que un programa fuente debe de pasar antes de convertirse en un programa ejecutable, los pasos son los siguiente

Analizador léxico
El analizador léxico o lexicográfico (Scanner en inglés) es la primera etapa del proceso de compilación, el cual se encarga de dividir el programa en Tokens, los cuales, según una tabla de símbolos definida por el mismo lenguaje. De esta forma cada token del programa es clasificado según su significado para ser procesados en la segunda etapa del proceso de compilación.

Analizador sintáctico
El analizador sintáctico es la segunda fase del proceso de compilación y tiene como finalidad la generación de un Árbol sintáctico, el cual no es más que una estructura de datos compleja que permite representar de una forma más simple al programa fuente.

Analizador semántico

El analizador semántico es el último paso antes de empezar a compilar realmente el código, prepara el programa para ser compilado. El analizador semántico parte del árbol sintáctico abstracto y tiene la finalidad de validar los puntos más finos del programa, como, por ejemplo, validar compatibilidad en tipos de datos, que la variable utilizada en una instrucción este previamente declara o que estén dentro del contexto, si implementamos una interface que todos los métodos estén definidos, etc. El analizador semántico es el que analiza que todo el programa tenga un significado exacto y que este no pueda fallar en tiempo de ejecución

- **Programa**

Es una secuencia de instrucciones, escritas para realizar una tarea específica en un computador. Este dispositivo requiere programas para funcionar, por lo general, ejecutando las instrucciones del programa en un procesador central. El programa tiene un formato ejecutable que la computadora puede utilizar directamente para ejecutar las instrucciones. El mismo programa en su formato de código fuente legible para humanos, del cual se derivan los programas ejecutables (por ejemplo, compilados), le permite a un programador estudiar y desarrollar sus algoritmos. Una colección de programas de computadora y datos relacionados se conoce como *software*.

Generalmente, el código fuente lo escriben profesionales conocidos como programadores de computadora. Este código se escribe en un lenguaje de programación que sigue uno de los siguientes dos paradigmas: imperativo (como si el programador diera órdenes concretas) o declarativo (se describe el resultado final deseado en lugar de todos los pasos a seguir), y que posteriormente puede ser convertido en un archivo ejecutable (usualmente llamado un programa ejecutable o un binario) por un compilador y más tarde ejecutado por una unidad central de procesamiento. Por otra parte, los programas de computadora se pueden ejecutar con la ayuda de un intérprete, o pueden ser empotrados directamente en *hardware*.

Un programa es un conjunto de pasos lógicos escritos en un lenguaje de programación que nos permite realizar una tarea específica. El programa suele contar con una interfaz de usuario, es decir, un medio visual mediante el cual interactuamos con la aplicación. Algunos ejemplos son la calculadora, el navegador de internet, un teclado en pantalla para el celular, etc.

Hoy encontramos programas o aplicaciones que pueden ejecutarse en una computadora, notebooks, tablets y celulares. Estas aplicaciones pueden ser escritas en diferentes lenguajes de programación. Como ejemplos encontramos C, Java, PHP, Python, entre otros. Estos programas corren sobre un sistema operativo, por ejemplo, Windows, Linux, Mac OS y Android entre otros.

Los programas para poder correr se deben cargar en la memoria, el responsable de esta tarea es el sistema operativo. Un programa puede diseñarse para una computadora o para otro tipo de dispositivos, pero su programación suele realizarse en una computadora utilizando un entorno de desarrollo integrado (en inglés IDE). Este programa cuenta con herramientas que permiten convertir nuestro código en un programa funcional. Estás herramientas son el compilador, el “linker” y el depurador (debugger).

Existen otras herramientas que facilitan nuestro trabajo, por ejemplo, para documentar o llevar registro de lo que hacemos (doxygen), para compartir nuestro trabajo y realizarlo en forma colaborativa (SVN / GIT). De esta forma, un equipo de trabajo puede desarrollar diferentes partes de un programa y luego integrarlas en forma más simple.

- ## Parches

En informática, un parche consta de cambios que se aplican a un programa, ya sea para corregir errores, agregarle alguna funcionalidad, actualizarlo, etc. Si bien los parches suelen ser desarrollados por programadores ajenos a los autores iniciales del proyecto, esto no siempre es así. Un parche puede ser aplicado tanto a un binario ejecutable como al código fuente de cualquier tipo de programa, incluso, un sistema operativo o una aplicación para dispositivos móviles.

Tipos de parches

- **Código fuente**: usado, sobre todo, en los softwares libres. Es un archivo de texto que indica las modificaciones necesarias en el código fuente del programa.
- **De traducción**: modifica el idioma definido del programa.
- **De depuración**: repara errores de programación o *bugs* que no se detectaron durante el desarrollo del programa. Cuando un programa puede contener este tipo de errores aparece la denominación beta.
- **De seguridad**: normalmente no modifican las funciones del programa y surgen para, como su nombre indica, solucionar problemas en cuanto a la seguridad. Son muy frecuentes en las aplicaciones que interaccionan en Internet.
- **De actualización**: se utiliza para mejorar algoritmos, añadir y optimizar el programa, etcétera. Es decir, incorpora nuevos métodos.

- **API**

Es un conjunto de definiciones y protocolos que se utiliza para desarrollar e integrar el software de las aplicaciones. API significa interfaz de programación de aplicaciones. Estas permiten que sus productos y servicios se comuniquen con otros, sin necesidad de saber cómo están implementados. Esto simplifica el desarrollo de las aplicaciones y permite ahorrar tiempo y dinero. Las API le otorgan flexibilidad; simplifican el diseño, la administración y el uso de las aplicaciones, y proporcionan oportunidades de innovación, lo cual es ideal al momento de diseñar herramientas y productos nuevos (o de gestionar los actuales).

A veces, las API se consideran como contratos, con documentación que representa un acuerdo entre las partes: si una de las partes envía una solicitud remota con cierta estructura en particular, esa misma estructura determinará cómo responderá el software de la otra parte.

Debido a que simplifican la forma en que los desarrolladores integran los elementos de las aplicaciones nuevas en una arquitectura actual, las API permiten la colaboración entre el equipo comercial y el de TI. Las necesidades comerciales suelen cambiar rápidamente en respuesta a los mercados digitales en constante cambio, donde la competencia puede modificar un sector entero con una aplicación nueva. Para seguir siendo competitivos, es importante admitir la implementación y el desarrollo rápidos de servicios innovadores. El desarrollo de aplicaciones nativas de la nube es una forma identificable de aumentar la velocidad de desarrollo y se basa en la conexión de una arquitectura de aplicaciones de microservicios a través de las API.

Las API son un medio simplificado para conectar su propia infraestructura a través del desarrollo de aplicaciones nativas de la nube, pero también le permiten compartir sus datos con clientes y otros usuarios externos. Las API públicas representan un valor comercial único porque simplifican y amplían la forma en que se conecta con sus partners y, además, pueden rentabilizar sus datos (un ejemplo conocido es la API de Google Maps).

- **Analizador PEG**

En informática, una gramática de expresión sintáctica (PEG) es un tipo de gramática formal analítica, es decir, describe un lenguaje formal en términos de un conjunto de reglas para reconocer cadenas en el lenguaje. El formalismo fue introducido por Bryan Ford en 2004 [1] y está estrechamente relacionado con la familia de lenguajes de análisis de arriba hacia abajo introducidos a principios de la década de 1970. Sintácticamente, los PEG también se parecen a las gramáticas libres de contexto (CFG), pero tienen una interpretación diferente: el operador de elección selecciona la primera coincidencia en PEG, mientras que es ambiguo en CFG. Esto está más cerca de cómo el reconocimiento de cuerdas tiende a realizarse en la práctica.

A diferencia de los CFG, los PEG no pueden ser ambiguos; si una cadena analiza, tiene exactamente un árbol de análisis válido. Se conjetura que existen lenguajes libres de contexto que no pueden ser reconocidos por un PEG, pero esto aún no está probado.

Los PEG son adecuados para analizar lenguajes informáticos (y lenguajes humanos artificiales como Lojban), pero no lenguajes naturales donde el rendimiento de los algoritmos PEG es comparable a los algoritmos CFG generales como el algoritmo Earley.

- **LLVM**

Es una infraestructura para desarrollar compiladores, escrita a su vez en el lenguaje de programación C++, que está diseñada para optimizar el tiempo de compilación, el tiempo de enlazado, el tiempo de ejecución y el "tiempo ocioso" en cualquier lenguaje de programación que el usuario quiera definir. Implementado originalmente para compilar C y C++, el diseño agnóstico de LLVM con respecto al lenguaje, y el éxito del proyecto han engendrado una amplia variedad de lenguajes, incluyendo Objective-C, Fortran, Ada, Haskell, bytecode de Java, Python, Ruby, ActionScript, GLSL, Clang, Rust, Gambas y otros.

El proyecto LLVM comenzó en 2000 en la Universidad de Illinois en Urbana-Champaign, bajo la dirección de Vikram Adve y Chris Lattner. LLVM fue desarrollado inicialmente bajo la Licencia de código abierto de la Universidad de Illinois, una licencia de tipo BSD. En 2005, Apple Inc. contrató a Lattner y formó un equipo para trabajar en el sistema de LLVM para varios usos dentro de los sistemas de desarrollo de Apple.[1] LLVM es parte integrante de las últimas herramientas de desarrollo de Apple para Mac OS X e iOS.[2]

El nombre "LLVM" era en principio las iniciales de "Low Level Virtual Machine", pero esta denominación causó una confusión ampliamente difundida, puesto que las máquinas virtuales son solo una de las muchas cosas que se pueden construir con LLVM. Cuando la extensión del proyecto se amplió incluso más, LLVM se convirtió en un proyecto paraguas que incluye una multiplicidad de otros compiladores y tecnologías de bajo nivel, haciendo el nombre aún menos adecuado. Por tanto, el proyecto abandonó[3] las iniciales. Actualmente, LLVM es una "marca" que se aplica al proyecto paraguas, la representación intermedia LLVM, el depurador LLVM, la biblioteca estándar de C++ definida por LLVM, etc.

- **IR**

El registro de instrucción IR es un registro de la unidad de control de la CPU en donde se almacena la instrucción que se está ejecutando. En los procesadores simples cada instrucción a ser ejecutada es cargada en el registro de la instrucción que la contiene mientras se es decodificada, preparada y al final ejecutada, un proceso que puede tomar varios pasos. Los procesadores más complejos usan una tubería de registros de instrucción donde cada etapa de la tubería hace parte del trabajo, decodificación, preparación, o ejecución, y después pasa el resultado a la siguiente etapa para realizar el siguiente paso hasta que la instrucción es procesada totalmente. Esto funciona como una línea de ensamblaje en donde en cada etapa se hace un trabajo parcial, y luego se pasa a la siguiente etapa para continuar con la fabricación del producto. Los procesadores modernos pueden incluso hacer algunos de los pasos de fuera de orden ya que la decodificación de varias instrucciones se hace en paralelo.

Decodificar el registro de instrucción incluye la determinación de la instrucción, también determinar dónde están sus operandos en memoria, leer los operandos desde la memoria, asignar recursos del procesador para ejecutar el comando (en procesadores super escalar), etc.

En informática, el registro de instrucción (IR) o el registro de instrucción actual (CIR) es la parte de la unidad de control de una CPU que contiene la instrucción que se está ejecutando o decodificando actualmente. [1] En los procesadores simples, cada instrucción a ejecutar se carga en el registro de instrucciones, que la mantiene mientras se decodifica, prepara y finalmente se ejecuta, lo que puede tomar varios pasos.

Algunos de los procesadores complicados utilizan una canalización de registros de instrucciones donde cada etapa de la canalización hace parte de la decodificación, preparación o ejecución y luego la pasa a la siguiente etapa para su paso. Los procesadores modernos incluso pueden realizar algunos de los pasos fuera de orden, ya que la decodificación de varias instrucciones se realiza en paralelo.

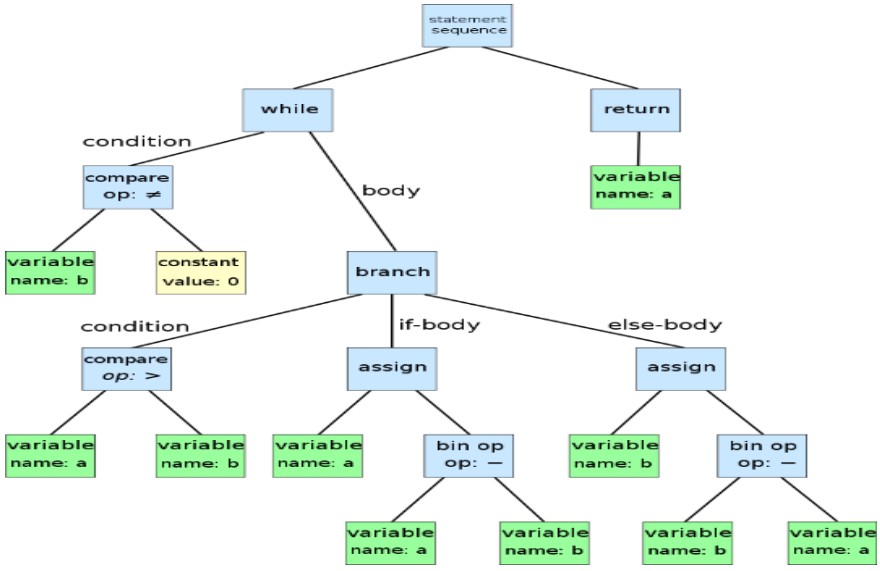
Decodificar el código de operación en el registro de instrucciones incluye determinar la instrucción, determinar dónde están sus operandos en la memoria, recuperar los operandos de la memoria, asignar recursos del procesador para ejecutar el comando (en procesadores súper escalares), etc. La salida del IR está disponible para los circuitos de control, que generan las señales de temporización que controlan los diversos elementos de procesamiento involucrados en la ejecución de la instrucción.

En el ciclo de instrucción, la instrucción se carga en el registro de instrucción después de que el procesador la recupera de la ubicación de memoria señalada por el contador del programa.

- **AST**

En lenguajes formales y lingüística computacional, un árbol de sintaxis abstracta (AST), o simplemente un árbol de sintaxis, es una representación de árbol de la estructura sintáctica simplificada del código fuente escrito en cierto lenguaje de programación. Cada nodo del árbol denota una construcción que ocurre en el código fuente. La sintaxis es abstracta en el sentido que no representa cada detalle que aparezca en la sintaxis verdadera. Por ejemplo, el agrupamiento de los paréntesis está implícito en la estructura arborescente, y una construcción sintáctica tal como IF condición THEN puede ser denotada por un solo nodo con dos ramas.

Esto hace a los árboles de sintaxis abstracta diferentes de los árboles de sintaxis concreta, llamados tradicionalmente árboles de par ser, que son a menudo construidos por la parte par ser de la traducción del código fuente y el proceso de compilación (a pesar quizás de un nombramiento no intuitivo). Una vez construido, información adicional es agregada al AST por procesamiento subsecuente, ej., análisis semántico.



- **Dialecto**

Es un lenguaje formal que comprende un conjunto de cadenas que producen varios tipos de salida de código de máquina. Los lenguajes de programación son un tipo de lenguaje informático y se utilizan en la programación informática para implementar algoritmos.

La mayoría de los lenguajes de programación constan de instrucciones para computadoras. Hay máquinas programables que utilizan un conjunto de instrucciones específicas, en lugar de lenguajes de programación generales. Desde principios del siglo XIX, se han utilizado programas para dirigir el comportamiento de máquinas como telares Jacquard, cajas de música y pianos. Los programas para estas máquinas (como los pergaminos de un piano) no produjeron un comportamiento diferente en respuesta a diferentes entradas o condiciones.

Se han creado miles de lenguajes de programación diferentes y cada año se crean más. Muchos lenguajes de programación están escritos en forma imperativa (es decir, como una secuencia de operaciones a realizar) mientras que otros lenguajes usan la forma declarativa (es decir, se especifica el resultado deseado, no cómo lograrlo).

La descripción de un lenguaje de programación generalmente se divide en dos componentes de sintaxis (forma) y semántica (significado). Algunos lenguajes están definidos por un documento de especificación (por ejemplo, el lenguaje de programación C está especificado por un estándar ISO) mientras que otros lenguajes (como Perl) tienen una implementación dominante que se trata como referencia. Algunos lenguajes tienen ambos, el lenguaje básico definido por un estándar y las extensiones tomadas de la implementación dominante son comunes.

Un dialecto de programación es una variación (relativamente pequeña) o una extensión de un Lenguaje de programación que no cambia su naturaleza intrínseca.

En lenguajes como Scheme o Forth, las normas o estándares se puede considerar insuficientes, inadecuados o ilegítimos, incluso por los implementadores, por lo que a menudo se desvían de la norma o del estándar, creando un nuevo dialecto. En otros casos, un dialecto se crea para su uso en algunos Lenguaje específico del dominio, a menudo un subconjunto.

En el mundo de Lisp, la mayoría de los lenguajes que utilizan la expresión básica Expresión S y una semántica similar a Lisp se consideran dialectos de Lisp, a pesar de que varían ampliamente, como lo hacen, por ejemplo, Bigloo Scheme, newLISP o Emacs Lisp.1. Es muy común que un lenguaje llegue a tener varios dialectos, hasta el punto en que puede llegar a ser bastante difícil para un programador sin experiencia encontrar la documentación necesaria. Otro ejemplo de un lenguaje con muchos dialectos es BASIC.

- **MLIR**

Es un proyecto que define una representación intermedia (IR) común que unifica la infraestructura necesaria para ejecutar modelos de aprendizaje automático de alto rendimiento en Tensor Flow y en marcos de trabajo de AA similares.

Este proyecto incluye la aplicación de técnicas de HPC (computación de alto rendimiento), junto con la integración de algoritmos de búsqueda, como el aprendizaje por refuerzo. El objetivo de MLIR es reducir el costo para incorporar hardware nuevo y mejorar la usabilidad para los usuarios de Tensor Flow.

MLIR tiene algunas propiedades interesantes (forma SSA, operaciones de dialecto, regiones) que hacen que el trabajo sea mucho más fácil de cambiar. Pero lo que es más importante, el IR tiene un flujo de control explícito (CFG), que es importante para rastrear de dónde provienen las variables, pasan a través de todas las combinaciones de rutas.

Para inferir tipos y verificar la seguridad, esto es fundamental para asegurarse de que el código no pueda llegar a un estado desconocido a través de al menos una de las rutas posibles. Entonces, la razón principal por la que elegimos MLIR para ser nuestra representación es para que podamos hacer nuestra inferencia de tipo más fácilmente.
La segunda razón es que MLIR nos permite mezclar cualquier número de dialectos. Por lo tanto, podemos reducir el AST a una mezcla de dialecto de Verona y otros dialectos estándar, y los pases que solo pueden ver las operaciones de Verona ignorarán a los demás y viceversa.
También nos permite bajar parcialmente partes del dialecto a otros dialectos sin tener que convertir todo. Esto mantiene el código limpio (pasadas cortas y directas) y nos permite construir lentamente más información, sin tener que ejecutar una pasada de análisis enorme seguida de una pasada de transformación enorme, solo para perder información en el medio.
Un beneficio inesperado de MLIR fue que tiene soporte nativo para operaciones opacas, es decir. operaciones similares a llamadas a funciones que no necesitan definirse en ninguna parte.

MLIR no tiene una representación de cadena nativa y no hay una forma sensata de representar todos los tipos de cadenas en los tipos existentes.

Verona

- **Mutación de concurrencia y seguridad de la memoria.**

En el proyecto Verona, los investigadores creen que abandonar la mutación concurrente es un paso necesario en la gestión de memoria escalable. Al eliminar las mutaciones de concurrencia, los desarrolladores no pueden implementar la concurrencia en las bibliotecas. Por lo general, hay dos opciones, exponer "inseguro" para permitir que las bibliotecas inseguras implementen concurrencia (como Rust), o proporcionar un modelo de concurrencia para el lenguaje (como Pony).

El primero significa que el lenguaje de programación solo puede confiar en menos invariantes porque no puede entender cómo el código en los módulos no seguros proporciona concurrencia. Esto último significa que se necesita una increíble Historia de concurrencia, porque solo puede haber una Historia de concurrencia.

- **Propiedad compartida**

En Verona, los investigadores introdujeron un nuevo modelo de programación concurrente: propietario concurrente, o abreviatura para abreviar. Encapsula algunos conjuntos de recursos (como áreas de memoria), al tiempo que garantiza que cada subproceso de ejecución acceda a estos recursos.

En Verona, podemos envolver un objeto en cown para que sea concurrente.

```
// x is some isolated object graphvar c = cown(x)
// c is a cown that mediates access to x.
// We have lost direct access to x here
```

Una vez que el usuario envuelve un objeto en un cown, solo puede acceder a él programando. En Verona, este método se realiza con la palabra clave when.

```
when (var x = c) {// Access internas of cowns(c) usan ñame x in here ñame("Ello\n")}ñame("Goode\n")
```

- **Dividir**

Verona utiliza particiones y grupos de objetos como conceptos básicos de propiedad. El investigador no especificó la propiedad del objeto como una referencia y un objeto, pero lo resumió como una referencia correspondiente puede poseer una partición, y una partición es un grupo de objetos. Dentro de una partición, cualquier objeto puede referirse a cualquier otro objeto dentro de la partición. Pero los investigadores no tienen restricciones en la topología. Cuando la referencia de toda la partición desaparece, toda la partición se recicla.

En el tipo, iso (aislado) se usa para indicar que se trata de una referencia a una partición. Para mutable, use mut para representar y representar referencias variables pero invencibles en el tipo. Cuando mut se usa para un tipo de campo, la referencia apunta a la misma partición que el objeto de campo. Cuando se usa mut para los tipos de parámetros, la referencia apunta a un objeto en una partición desconocida. Esto se basa en un tipo en Rust.

Al asignar un objeto, especifique si debe estar en su propia partición:

```
var x = new Node;
```

o en la misma partición que otro objeto:

```
var y = new Node in x
```

Las particiones se pueden anidar y formar un árbol de partición, donde la partición raíz está en la pila o en cowns.

- **Prueba del sistema**

Inspirados en P y P #, el tiempo de ejecución y las pruebas del sistema de Verona están profundamente integrados en el momento del diseño. El modelo de concurrencia de Verona permite que todas las interacciones concurrentes tengan lugar en el tiempo de ejecución.

La aplicación básica fue diseñada originalmente para pruebas de tiempo de ejecución, pero los investigadores pensaron en un compilador alternativo para el lenguaje, porque ayudó a la prueba. Está construido por veronac-sys e interpeter-sys y requiere los siguientes parámetros adicionales:

--run-seed N --run-seed_upper N

Entonces

veronac-sys.exe --run testsuite/demo/run-pass/dining_phil.verona --run-seed 100 --run-seed_upper 200

Con el siguiente comando, puede ejecutar 100 muestras intercaladas (un método de aceleración de memoria).

veronac-sys.exe --run testsuite/demo/run-pass/dining_phil.verona --run-seed 100 --run-seed_upper 200

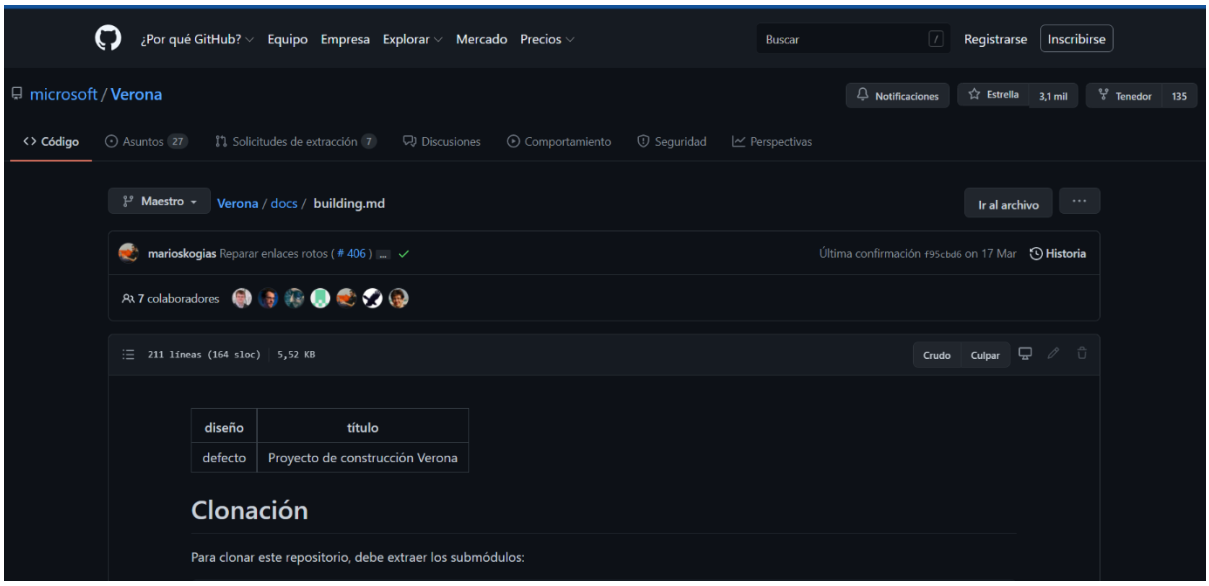
Según la introducción del proyecto, este proyecto espera desarrollar una mayor cooperación con la comunidad de investigación a través del código abierto. El proyecto aún se encuentra en sus primeras etapas y no afectará los proyectos de ingeniería utilizados por el propio Microsoft.

DESARROLLO

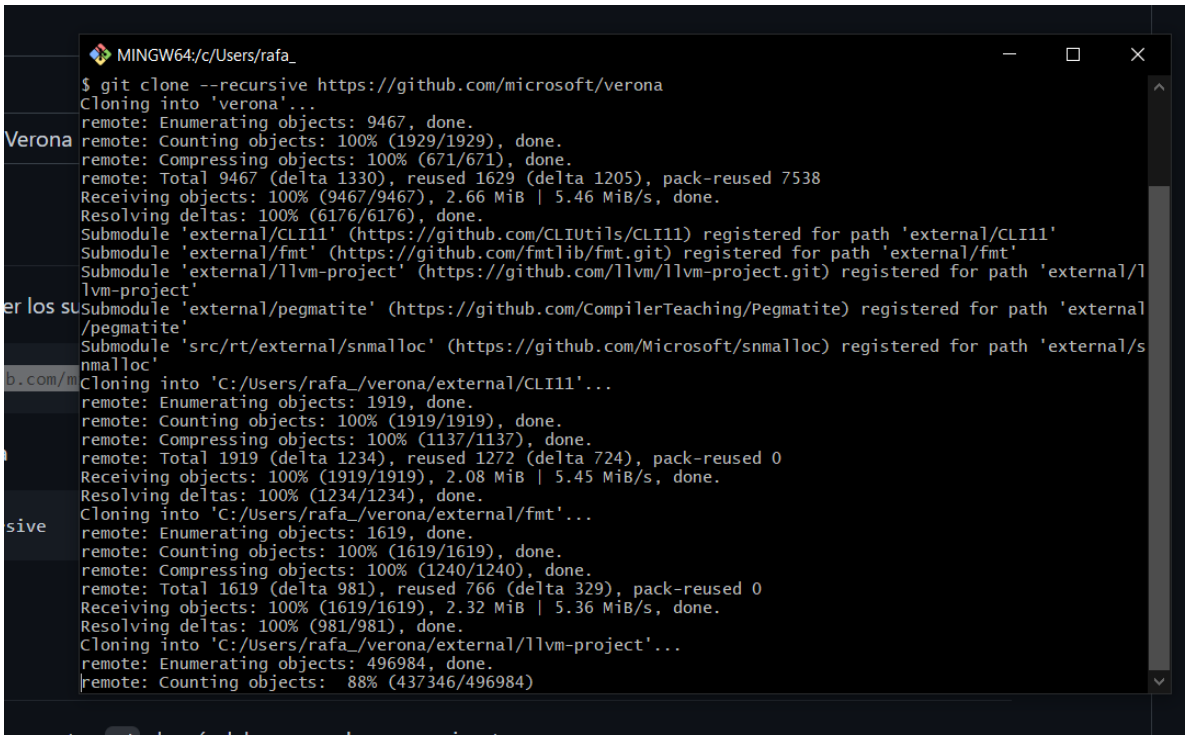
INSTALACION

1. Entramos en la página de Microsoft donde encontramos el repositorio de Verona para poder descargarlo.

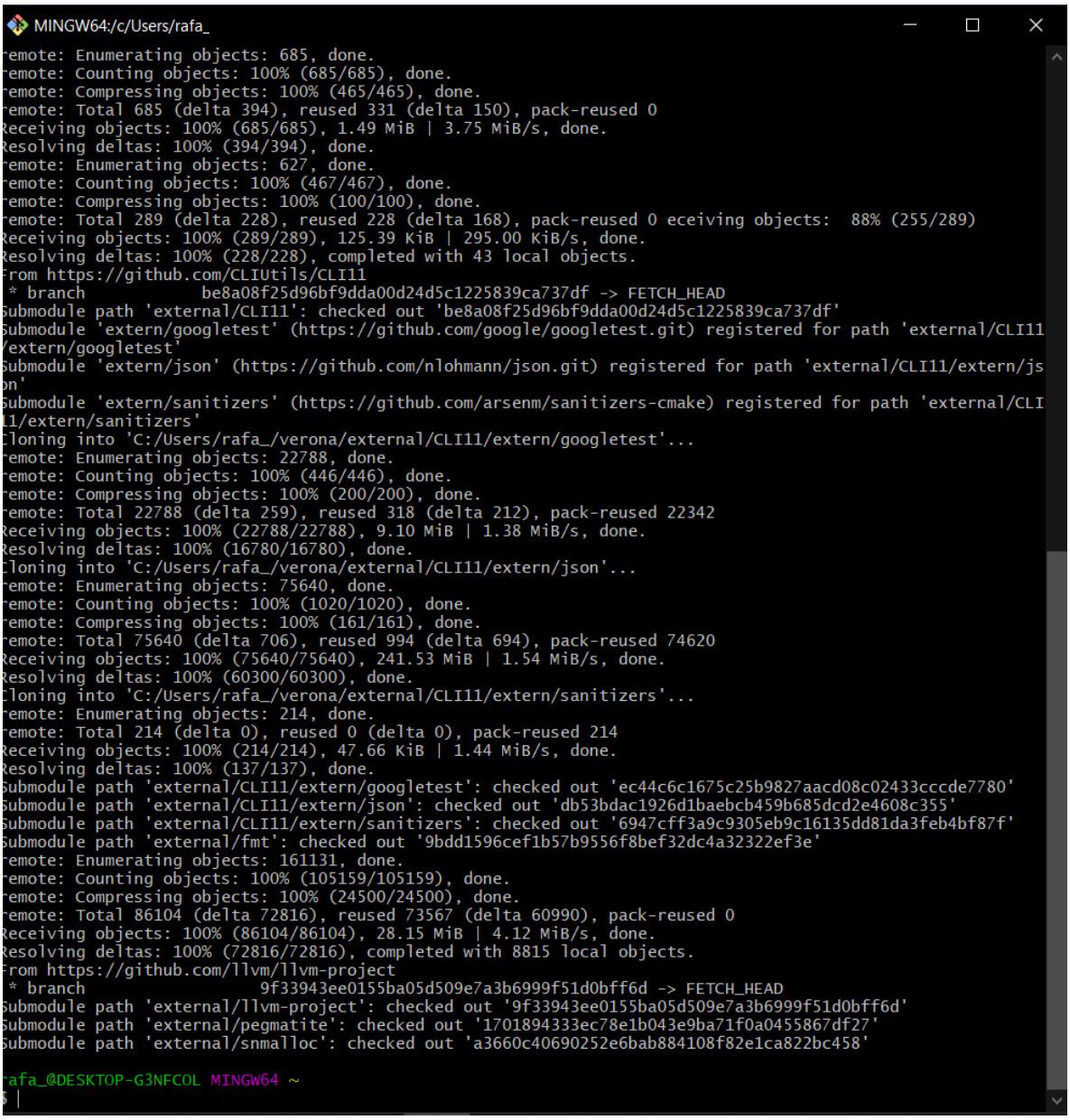
   https://github.com/microsoft/verona/blob/master/docs/building.md



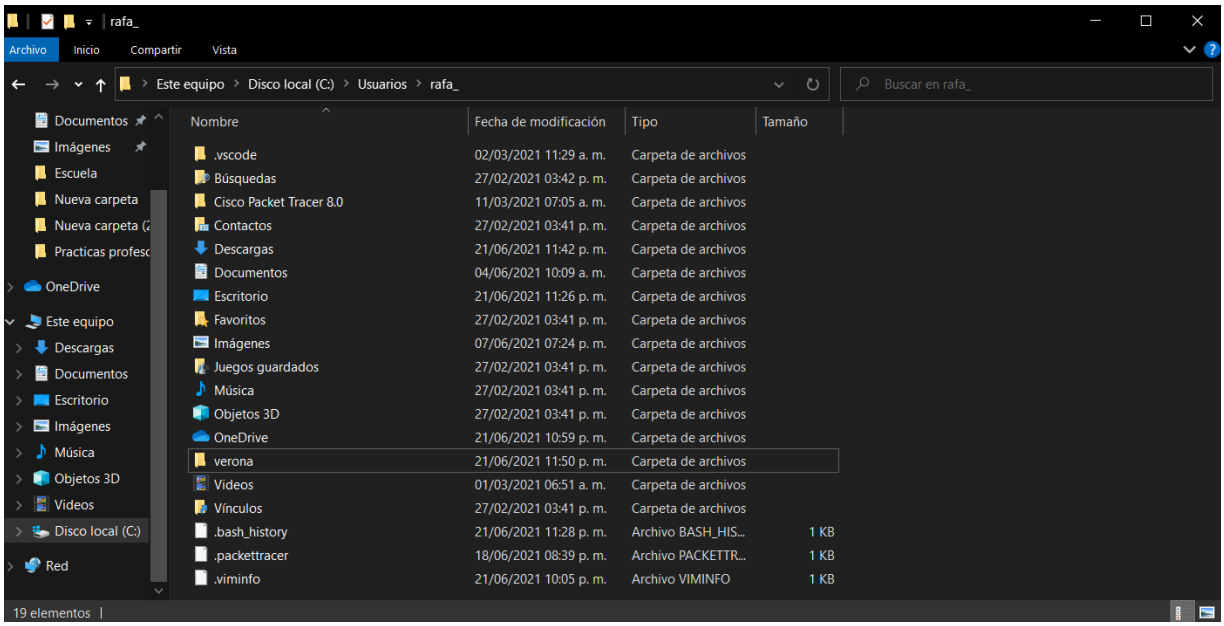
2. Clonamos el repositorio de github con ayuda de la terminal de gitbash
   git clone --recursive https://github.com/microsoft/verona



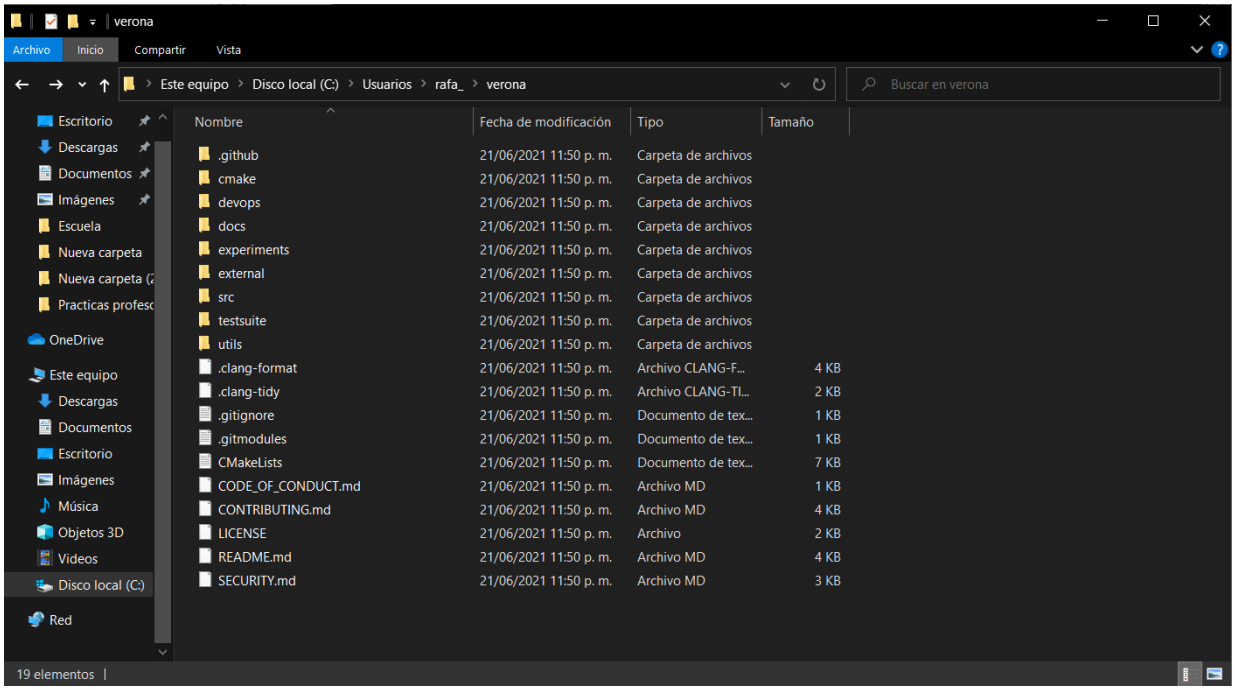
3. Cuando se acabe de clonar el repositorio no aparece lo siguiente.



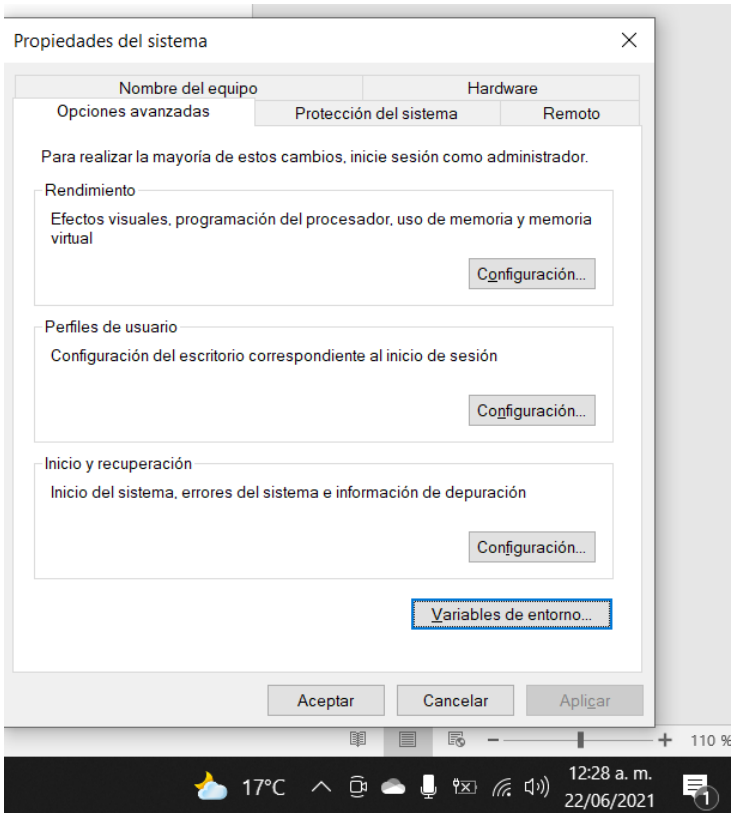
4. Nos va a crear la carpeta de Verona.

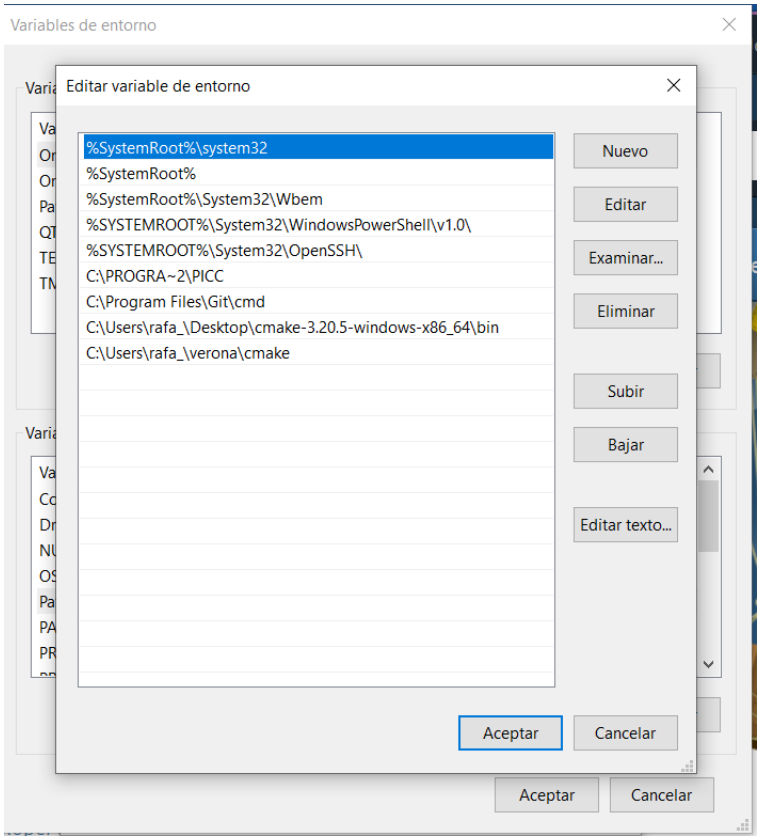


5. Y estos son los documentos que tenemos en la carpeta de Verona.

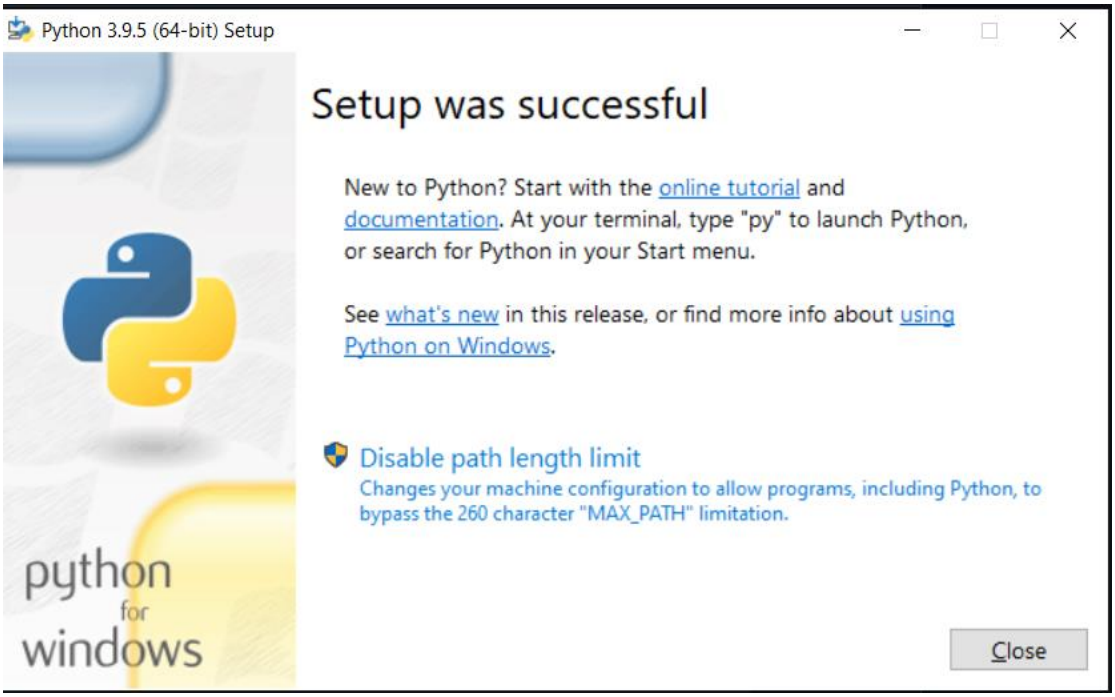


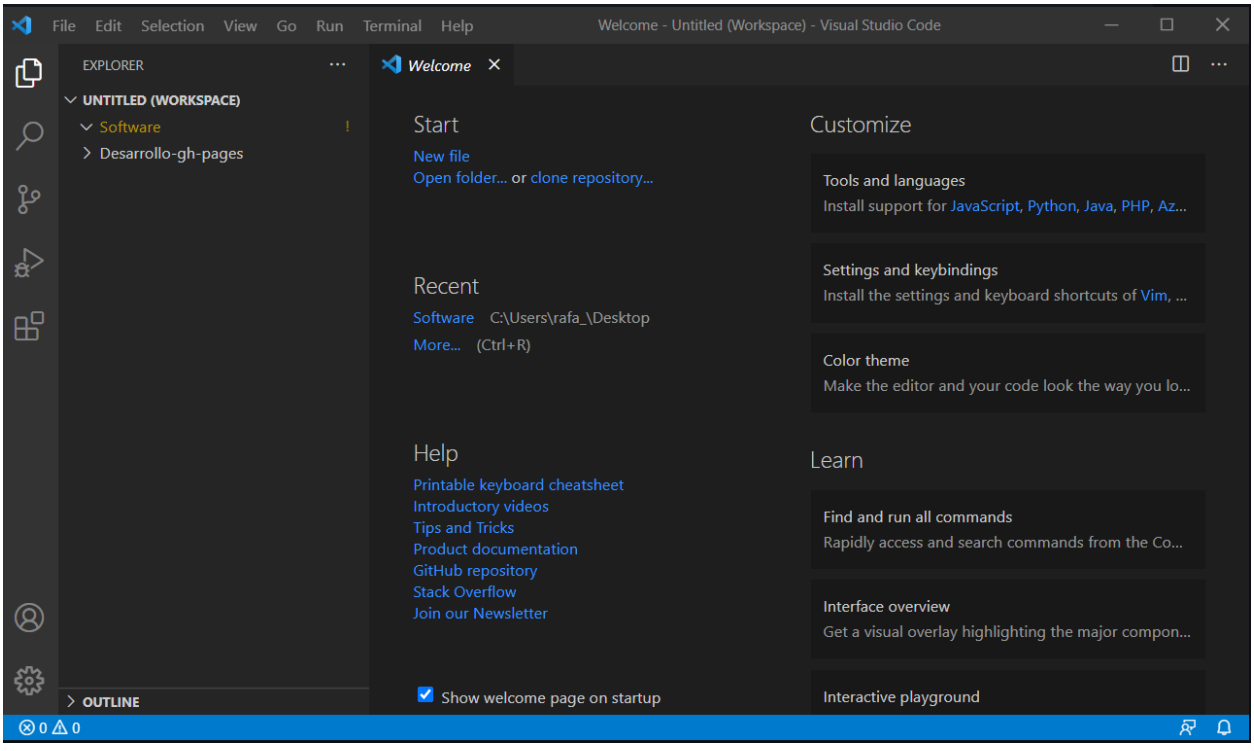
6. Debemos instalar CMake, entraremos a las Variables de entorno.



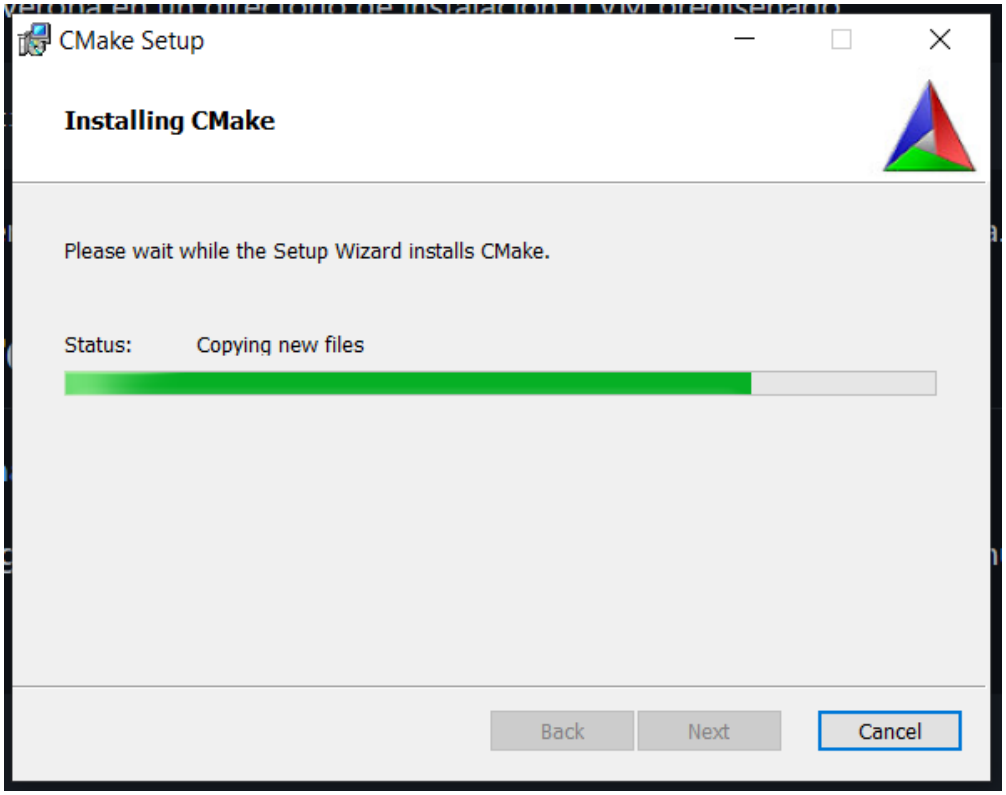


7. Se va a trabar con Visual Studio y Python 3.

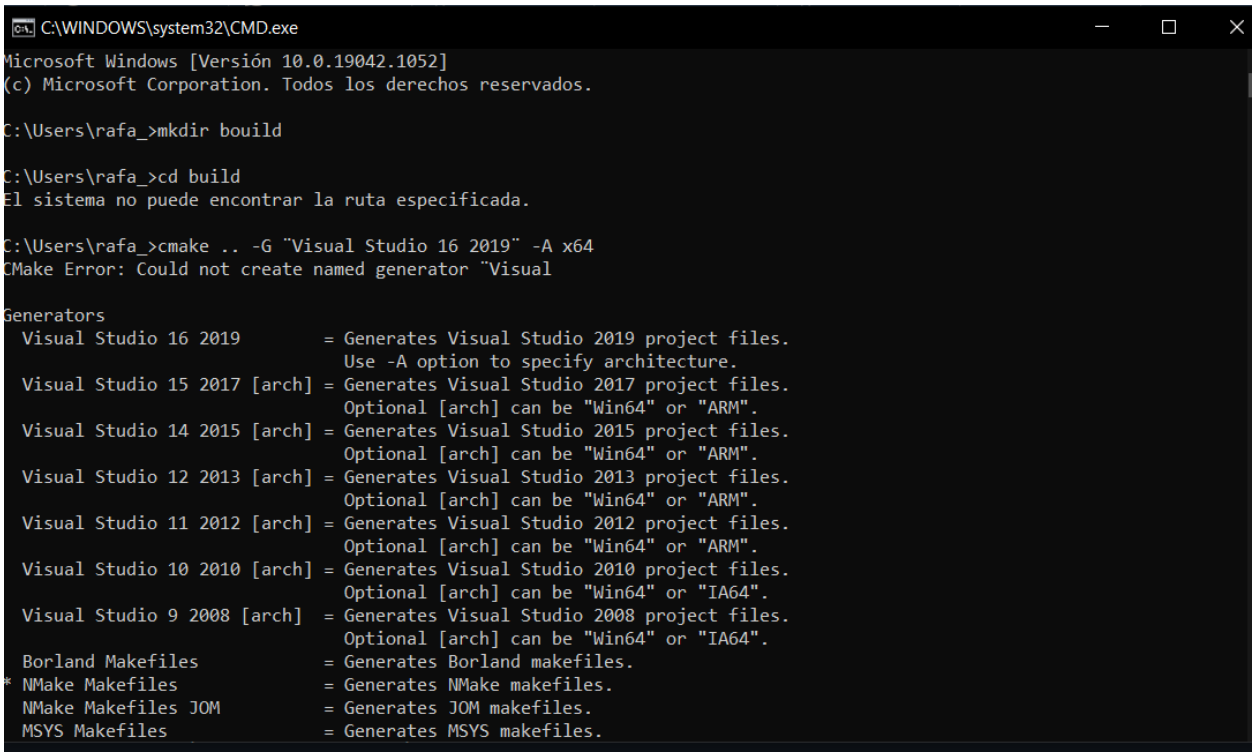




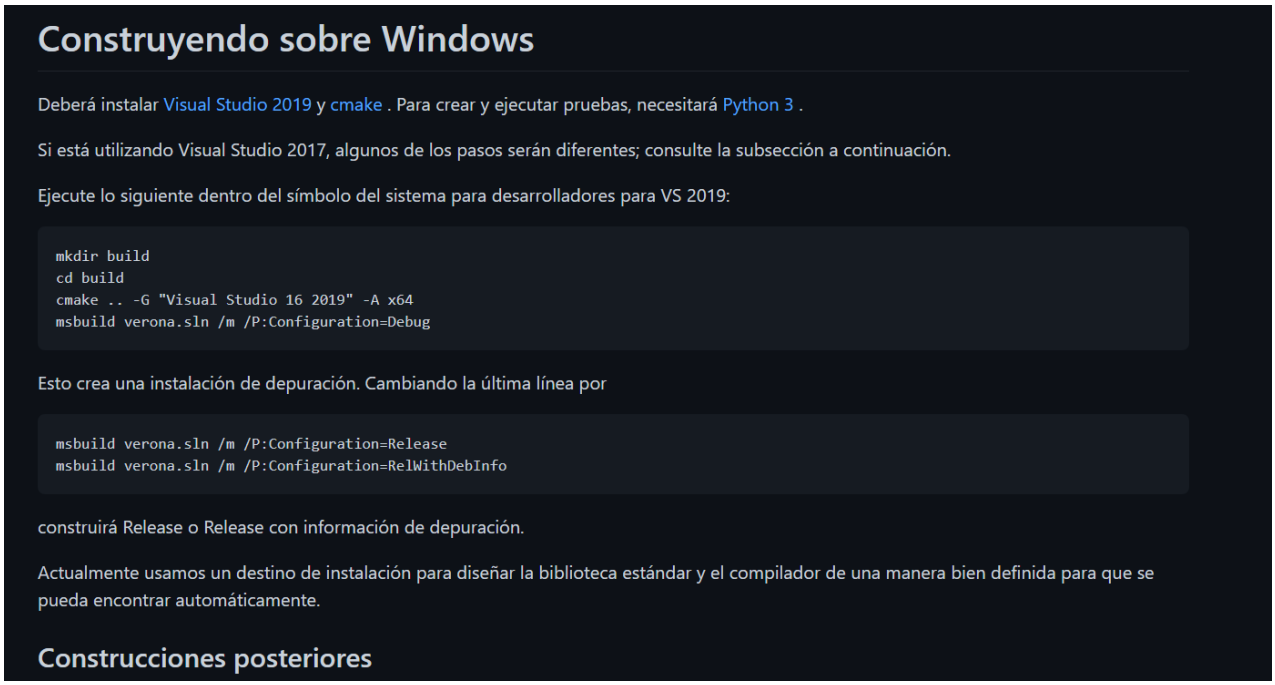
8. Se necesita instalar CMake para poderlo usar con Visual Studio.



9. Se configura desde línea de comando.



10. Lo utilizamos para construir en Windows con ayuda de VS.



Construyendo sobre Windows

Deberá instalar Visual Studio 2019 y cmake . Para crear y ejecutar pruebas, necesitará Python 3 .

Si está utilizando Visual Studio 2017, algunos de los pasos serán diferentes; consulte la subsección a continuación.

Ejecute lo siguiente dentro del símbolo del sistema para desarrolladores para VS 2019:

```
mkdir build
cd build
cmake .. -G "Visual Studio 16 2019" -A x64
msbuild verona.sln /m /P:Configuration=Debug
```

Esto crea una instalación de depuración. Cambiando la última línea por

```
msbuild verona.sln /m /P:Configuration=Release
msbuild verona.sln /m /P:Configuration=RelWithDebInfo
```

construirá Release o Release con información de depuración.

Actualmente usamos un destino de instalación para diseñar la biblioteca estándar y el compilador de una manera bien definida para que se pueda encontrar automáticamente.

Construcciones posteriores

11. Se siguen los pasos para las siguientes construcciones.

Construcciones posteriores

Para compilaciones posteriores, no es necesario volver a ejecutar `cmake`. Desde el `build` directorio, puede ejecutar

```
msbuild verona.sln /m
```

La configuración predeterminada es Debug.

Usando Visual Studio 2017

Si está utilizando Visual Studio 2017, deberá ejecutar comandos dentro del símbolo del sistema para desarrolladores para VS 2017. Además, el `cmake` comando es diferente:

```
cmake .. -G "Visual Studio 15 2017 Win64"
```

Construyendo sobre una plataforma similar a UNIX

Requisitos previos para construir en Linux

Estos pasos se probaron en Ubuntu 18.04.

Primero, necesitará instalar dependencias:

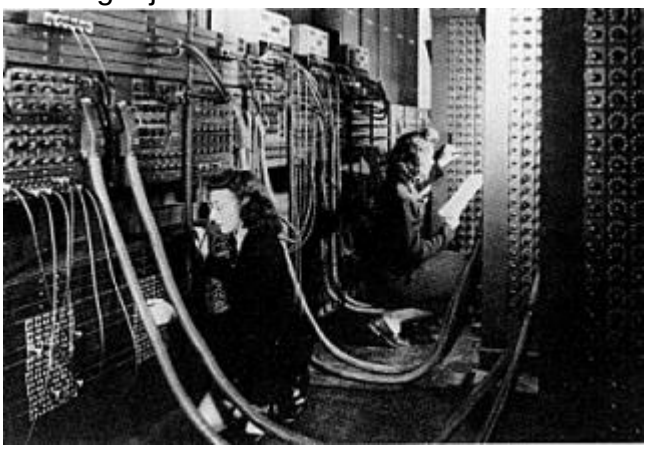
CODIGO ABIERTO

¿Qué significa que un lenguaje de programación sea de código abierto?
El código abierto es un modelo de desarrollo de software basado en la colaboración abierta. Se enfoca más en los beneficios prácticos que en cuestiones éticas o de libertad que tanto se destacan en el software libre. Para muchos el término «libre» hace referencia al hecho de adquirir un software de manera gratuita.

Teniendo esto en cuenta, un lenguaje de programación de código abierto es un lenguaje cuya definición y "funcionamiento" (a veces compilador) están disponible al público tanto para conocerlo como para realizar sus modificaciones y colaborar para su mejoramiento, crecimiento y mantenimiento por parte de la comunidad.

Ejemplos: C# Python Java Perl entre otros, el lenguaje no siempre es Open-source los IDE o compiladores

¿Si los lenguajes de programación se crean a partir de otros lenguajes de programación, ¿cómo se creó el primer lenguaje?

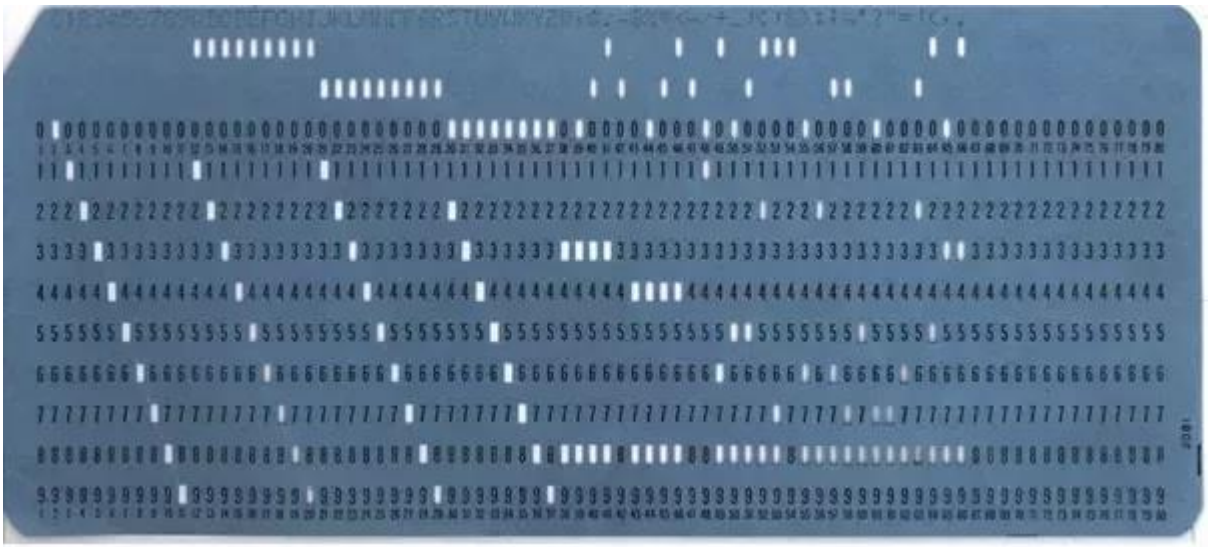


No es cierto que “los lenguajes de programación se creen a partir de otros lenguajes de programación”. No existe nada parecido a un enigma de “huevo-gallina” porque un lenguaje no crea otro lenguaje. Son las herramientas asociadas a los lenguajes de programación (compiladores, enlazadores, intérpretes, etc.) las que suelen crearse en otros lenguajes de programación.

En todo caso, antes de que existieran lenguajes de programación de alto nivel existían programadores. Programaban usando códigos numéricos que representaban instrucciones, registros, posiciones de memoria, constantes numéricas, caracteres, etc.

En las máquinas más antiguas, esto se hacía cableando directamente los circuitos de la máquina. La siguiente fotografía muestra a dos de las primeras programadoras de la historia cableando los circuitos de uno de los primeros ordenadores de propósito general del siglo XX; el famoso ENIAC:

Posteriormente (cuando se inventó el concepto de programa almacenado en memoria) se pasó a codificar los cableados en tarjetas perforadas o en cintas de papel:



Una vez perforada la electrónica requerida, una máquina llamada tabuladora cargaba esos programas en binario en la memoria de la máquina. En realidad, las tarjetas se venían usando ya desde antes de la existencia de los primeros ordenadores electrónicos (fueron aplicadas al cálculo por primera vez en 1890… e incluso ya se habían usado para otros fines como la codificación de patrones de telar en el siglo XVIII).

En todo caso, hacia finales de los 40 los programadores usaban directamente algo que se conoce como lenguaje máquina: esencialmente unos y ceros que representaban los códigos numéricos a los que antes hacíamos referencia.

El siguiente punto de referencia lo encontramos en 1947, cuando una programadora inglesa llamada Kathleen Booth, que trabajaba con John Von Newman, ideó el primer ensamblador de la historia.

Aunque este cambio no parece que suponga un gran avance (después de todo se limitaba a codificar con códigos alfanuméricos el lenguaje máquina puro: los números que representaban las operaciones, los registros, las direcciones en memoria direcciones, los desplazamientos, etc.) en realidad sí supuso un cambio importante en la forma en la que se programaba.

El paso de usar códigos binarios puros a ensamblador permitió especializar dos funciones distintas: la programación en ensamblador y la traducción a binario. Esto permitió que una persona se encargara de la lógica del programa y otra a traducir ese programa a código máquina.

Aun así, la programación era tediosa, se hacía manualmente y estaba sujeta a muchos errores.

Por eso algunos programadores empezaron a escribir procedimientos y herramientas que permitían interpretar expresiones de alto nivel. Uno de los primeros ejemplos lo ofreció el intérprete de Short Code (1949):

```
1. X3 =  (  X1 +  Y1 )  /  X1 * Y1   # Sustituir variables
2. X3 03 09 X1 07 Y1 02 04 X1   Y1   # Sustituir opers/paréntesis.
3.                    # ^
4.                    # Nótese que el producto se representa por
5.                    # juxtaposición y no tiene operador propio
6. 07Y10204X1Y1                 # Agrupar en palabras de 12 bytes
7. 0000X30309X1                 # y enviar al intérprete.
```

Hay que aclarar que varios de los pasos del anterior proceso se seguían haciendo de forma manual.

Poco después las terminales empezaron a introducirse y, las perforadoras y las tabuladoras se sustituyeron por teclados electrónicos, y los dispositivos de salida basados en papel impreso se sustituyeron por tubos de rayos catódicos en los que se imprimían letras y símbolos.

Este cambio también propició que se desarrollaran programas que se encargaban de traducir automáticamente los códigos de ensamblador a código binario ejecutable, así como la generalización de expresiones de alto nivel similares a las que hemos visto más arriba y la automatización de las llamadas a los intérpretes.

Y finalmente llegamos al primer lenguaje de alto nivel de la historia. Obviamente el propósito de su creador, John W. Backus, fue liberarse del tedioso ensamblador que requería gran cantidad de esfuerzo para programar:

- Cuál: Fortran.
- Dónde: Laboratorios de IBM.
- Quién: John W. Backus y un equipo de nueve programadores.
- Cuándo empezó el proyecto: 1953.
- Cuándo apareció el primer compilador: 1957.
- Para qué máquina: IBM 704.
- Cuánto esfuerzo fue necesario: 18 personas/año.

¿Por qué se considera que un software de código abierto es mejor que el software de código cerrado, y cuáles son los beneficios?

El esquema de código abierto tiene, como todo, sus ventajas y desventajas:

Algunas ventajas:

- El software, al tener expuesto el código fuente, es generalmente gratuito o de muy bajo costo para el usuario final.
- Participación y transparencia. El código abierto está disponible para todos y se puede participar, como usuario final, en cierto grado en las discusiones sobre la dirección general, las mejoras o correcciones del producto.
- Está continuamente evolucionando ya que la comunidad de programadores está agregando mejoras o corrigiendo errores de forma continua. Como desarrollador se puede modificar, mejorar o corregir el código y solicitar su inclusión en el código final.
- Tiende a ser un poco más confiable porque tiene mayor exposición a usuarios y desarrolladores por lo que los problemas salen a la luz con mayor rapidez y pueden ser corregidos por cualquiera.
- Flexibilidad. Se puede modificar o adaptar para integrar alguna característica o necesidad en particular.
- La dificultad y el tiempo invertido en desarrollo y mantenimiento se diluye entre toda la comunidad de desarrolladores del producto y no en un solo individuo o empresa.

Algunas desventajas:

- El producto puede ser abandonado por sus desarrolladores por diversas razones.
- Tienden a ser menos amigables o tener interfaces menos pulidas porque usualmente se enfocan más en la funcionalidad.
- El producto puede evolucionar y crecer en alguna dirección distinta a la original. En ocasiones los desarrolladores pueden dictar o influir más en la dirección del producto que los usuarios finales.
- En algunos proyectos el soporte depende totalmente de la comunidad de usuarios. No hay un departamento o área dedicada al soporte.
- Al tener el código fuente expuesto, alguna persona mal intencionada puede buscar y tratar de abusar de las fallas en el mismo, o incluso tratar de inyectar código malicioso directamente en el código fuente.

Espero que estos puntos de ayuden a tomar una mejor decisión para calificarlo como mejor o peor, pero depende de las necesidades y del punto de vista de cada uno: si se va iniciar un proyecto como desarrollador independiente o como empresa, si se va a utilizar software abierto en toda la empresa o si se va a utilizar de forma personal.

ADMINISTRACION DE MEMORIA

La memoria es un recurso importante que debe ser cuidadosamente gestionado. A todo programador le gustaría poder contar con una memoria infinitamente grande, infinitamente rápida y que fuese además no volátil, esto es, que no perdiese su contenido en ausencia de energía eléctrica. Pero al no poder contar con algunas de estas características han surgido técnicas y algoritmos capaces de administrar de una forma prima la memoria de nuestra computadora.

¿Qué es la Administración de Memoria?

Es una tarea realizada por el sistema operativo que consiste en gestionar la jerarquía de memoria, en cargar y descargar procesos en memoria principal para que sean ejecutados. Para ello el sistema operativo gestiona lo que se conoce como MMU o Unidad de Administración de Memoria, el cual es un dispositivo hardware que transforma las direcciones lógicas en físicas.

Su trabajo es seguir la pista de que partes de la memoria están en uso y cuáles no lo están, con el fin de poder asignar memoria a los procesos cuando la necesiten, y recuperar esa memoria cuando dejen de necesitarla, así como gestionar el intercambio entre memoria principal y el disco cuando la memoria principal resulte demasiado pequeña para contener a todos los procesos

Objetivos de la Gestión de Memoria

- Ofrecer a cada proceso un espacio lógico propio.
- Proporcionar protección entre los procesos.
- Permitir que los procesos compartan memoria.
- Maximizar el rendimiento del sistema.

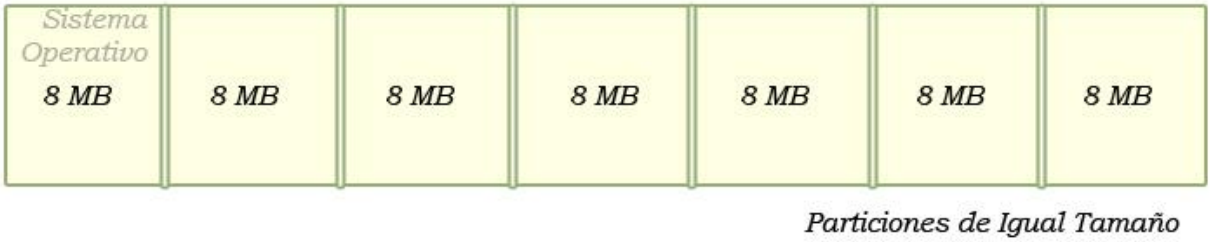
Requisitos de la Gestión de Memoria

- **Reubicación:** En un sistema multiprogramado la memoria se encuentra compartida por varios procesos, por lo tanto, los procesos deben ser cargados y descargados de memoria.
- **Protección:** En un sistema con multiprogramación es necesario proteger al sistema operativo y a los otros procesos de posibles accesos que se puedan realizar a sus espacios de direcciones.
- **Compartición:** En ciertas situaciones, bajo la supervisión y control del sistema operativo, puede ser provechoso que los procesos puedan compartir memoria.
- **Organización Lógica:** Tanto la memoria principal como la secundaria presentan una organización física similar, como un espacio de direcciones lineal y unidimensional. Debe existir una cierta correspondencia entre el sistema operativo y el hardware al tratar los datos y los programas de los usuarios de acuerdo a la estructura lógica que ellos presenten.

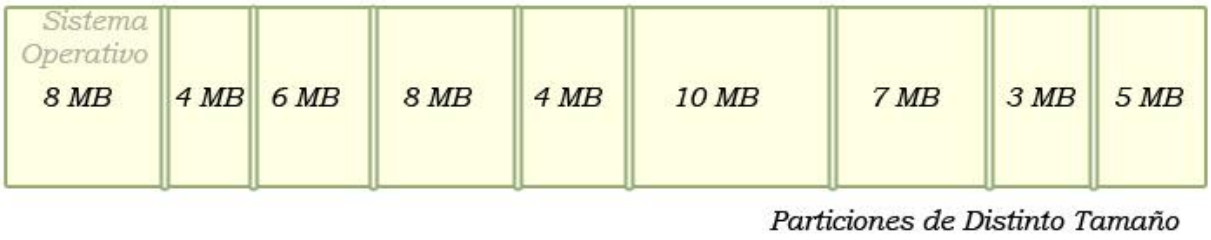
- **Organización Física:** Debe ser parte de la administración de memoria, la organización del flujo de información entre la memoria principal y la memoria secundaria.

Técnicas de Administración de Memoria (Asignación Contigua)

Ejemplo de particiones estéticas

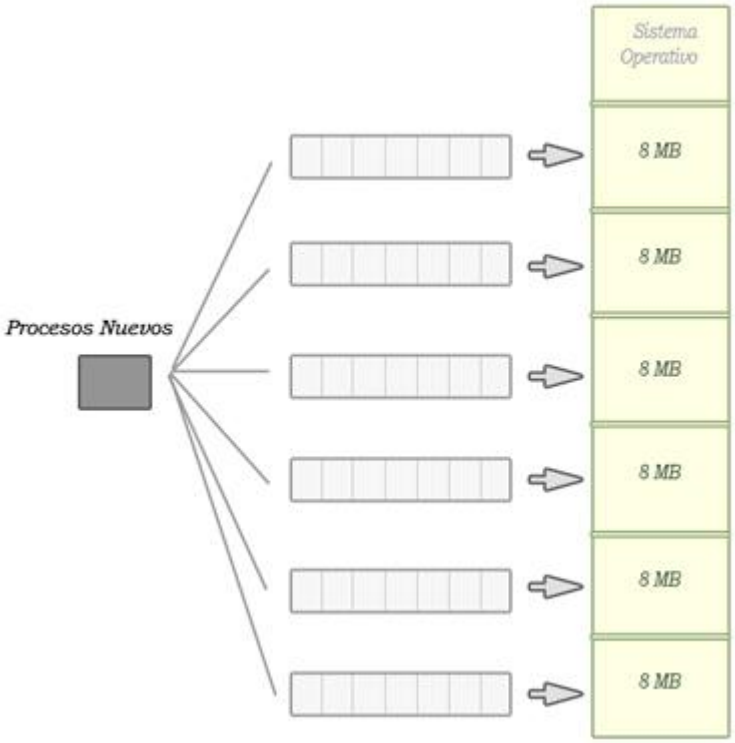


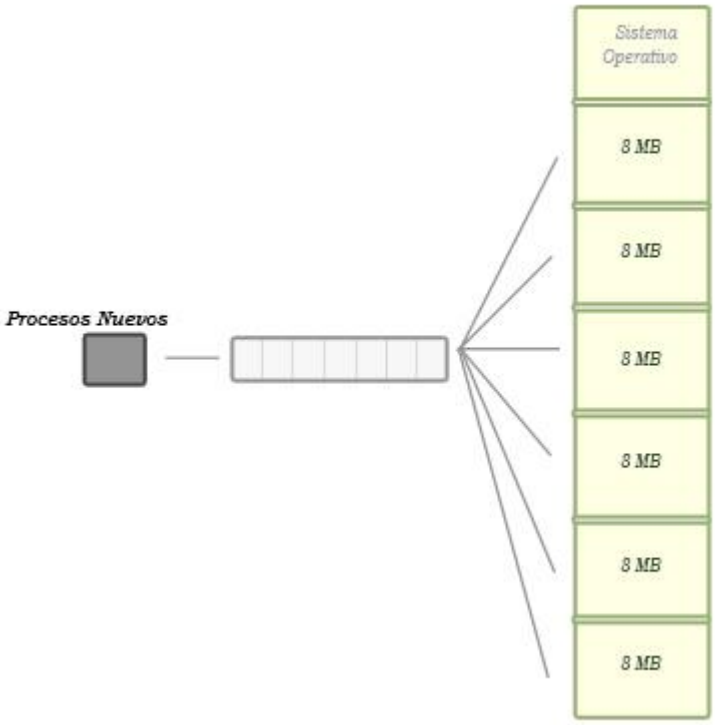
Particiones de Igual Tamaño



Particiones de Distinto Tamaño

Asignación de Memoria con particiones estéticas

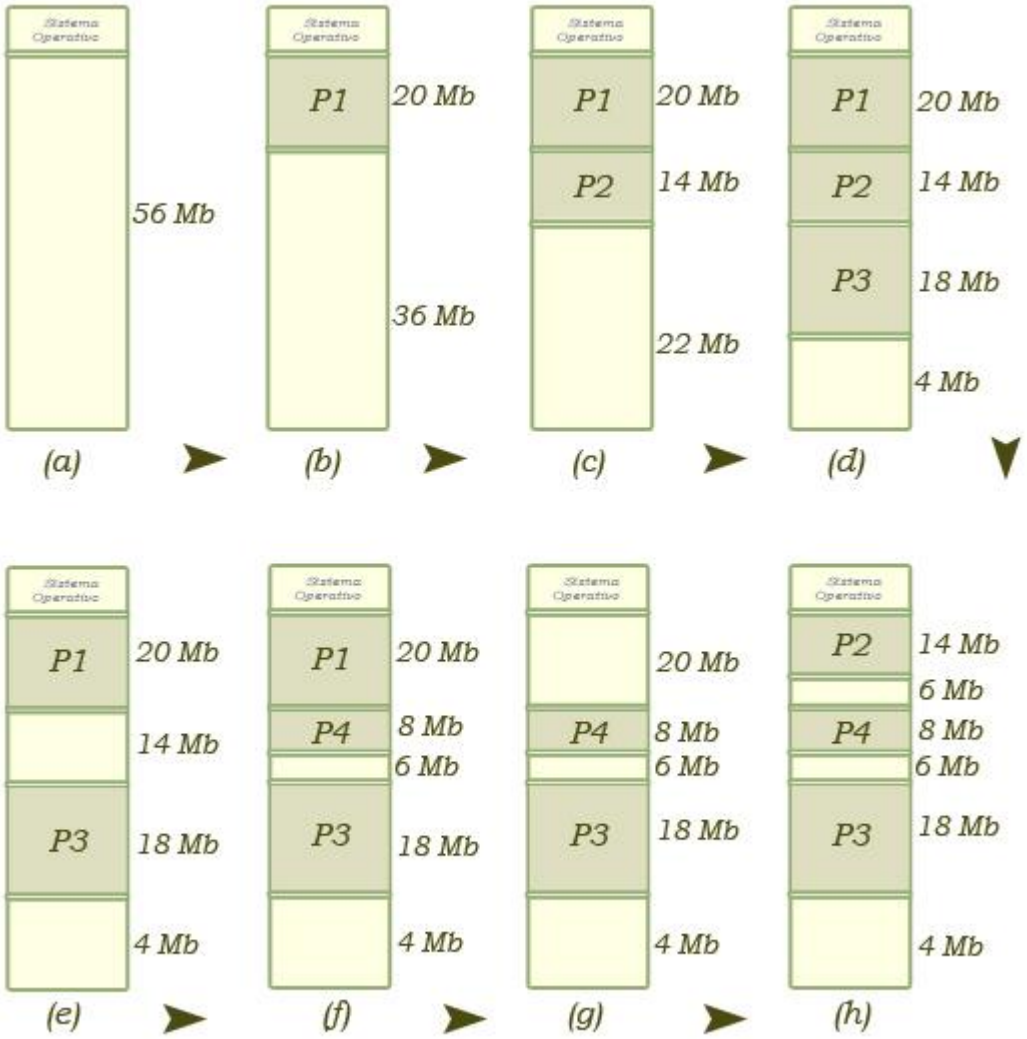




Con particiones estéticas surgen dos dificultades:

- Un programa puede ser demasiado grande para caber en una partición, por lo tanto, si el programa no se ha diseñado mediante superposición, simplemente no se puede ejecutar. De otro modo, podrán estar en memoria aquellos módulos del programa que se necesiten, pero se requerrá que estos módulos sean intercambiados a medida que la ejecución progresa.
- Se malgasta el espacio interno a cada partición cuando el bloque cargado es más pequeño, lo que se conoce como fragmentación interna. Es decir, cualquier proceso por pequeño que sea, ocupará una partición completa.

Ejemplo de particiones dinámicas



*En este ejemplo, partimos de la memoria libre completamente en **(a)**, luego se cargan "P1" [20 MB], **(b)**, "P2" [14 MB] **(c)** y "P3" [18 MB] **(d)**. Se libera "P2" **(e)** y se carga "P4" [8 MB] **(f)**, se libera "P1" **(g)** y se carga nuevamente "P2" [14 MB] **(h)**. Notemos como se van reestructurando las particiones en base al tamaño de los procesos que se van cargando, esto sucede por ser particiones dinámicas.*

La asignación de memoria en un esquema con particiones dinámicas, consiste en determinar en qué hueco ubicar un nuevo proceso. Para esto existen tres algoritmos: mejor ajuste, primer ajuste o próximo ajuste.

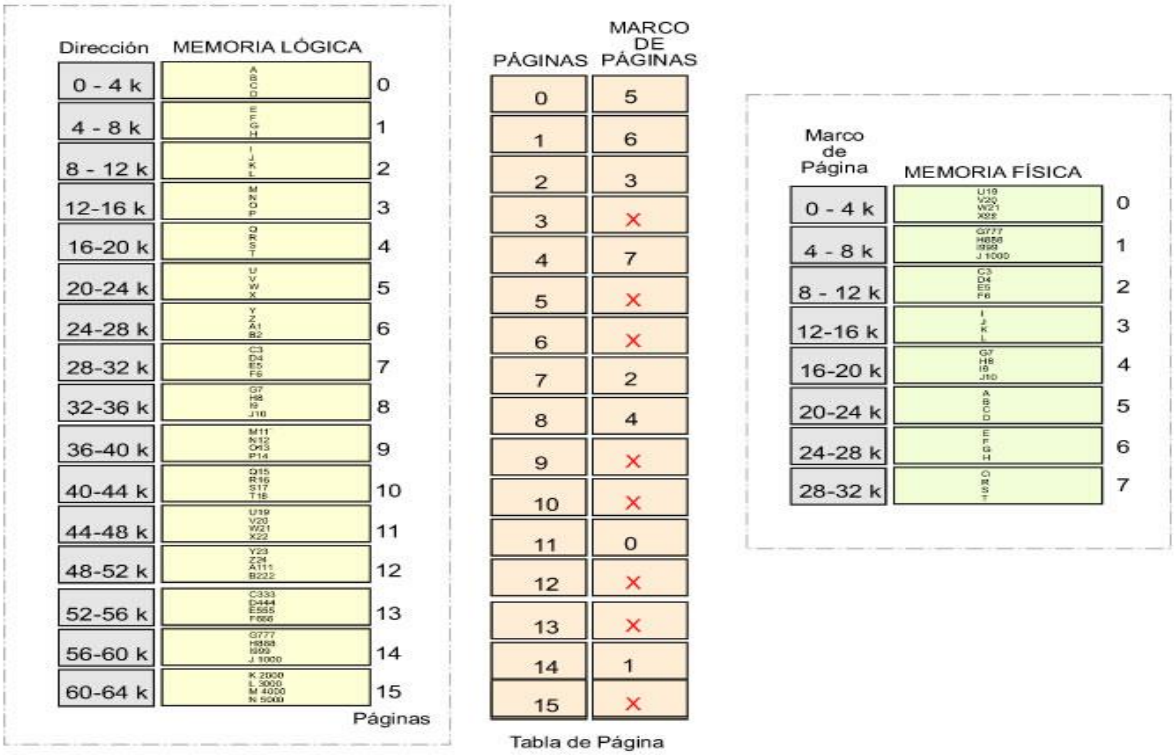
- **Mejor ajuste:** consiste en ubicar el proceso en el espacio de memoria que más se ajuste a su tamaño.
- **Primer ajuste:** consiste en ubicar el proceso en el primer hueco disponible, recorriendo desde el inicio de la memoria, cuyo tamaño sea suficiente para el proceso.
- **Próximo ajuste:** consiste en ubicar el siguiente hueco disponible, que sea suficientemente grande, a partir de la úllima asignación de memoria.

Con particiones dinámicas surgen las siguientes dificultades:

- Producto de la entrada y salida de procesos en la memoria, se van generando porciones cada vez más pequeñas de la memoria sin utilizar, lo que se conoce como fragmentación externa.
- Para solucionar este problema se debe recurrir a la compactación de la memoria de manera de eliminar los espacios (huecos) entre procesos. Esto significa que los procesos deben ser reubicados en memoria en forma dinámica.

Técnicas de Administración de Memoria (Asignación no Contigua)

Dentro de la asignación no contigua, la administración de memoria implementa técnicas como Paginación y Segmentación



Paginación

Esta técnica consiste en dividir la memoria en espacios de igual tamaño llamados páginas, en la memoria lógica y marcos de página en la memoria física. Mostramos una imagen de cómo se relacionan la memoria lógica con la memoria física a través de la Tabla de Páginas.

Segmentación

Consiste en dividir la memoria en espacios al igual que la Paginación, pero a diferencia de esta, la divide en espacios de diferente tamaño. Mostramos en la imagen la relación entre la memoria lógica y la física a través de la Tabla de Descriptores de Segmento (TDS).

Conclusiones

- Una de las tareas más complejas e importantes que lleva a cabo el sistema operativo es la de administrar la memoria.
- Todos los métodos aquí presentados, suponen que para ejecutar un proceso es necesario tenerlo completamente en memoria principal, ya sea en espacios contiguos o no.
- La paginación es un esquema similar al de las particiones estéticas, con la ventaja de que un proceso puede ser cargado en más de una partición y en espacios de memoria no contiguos, lo que reduce la fragmentación interna a la que se produce en la última página.
- La segmentación es un esquema similar al de las particiones dinámicas con ventajas similares a la paginación.
- Tanto en las particiones estéticas, como en la paginación, es necesario determinar el tamaño que tendrán los bloques de memoria.

CARACTERISTICAS DE LOS LENGUAJES RUST Y CYCLONE

RUST

Rust es un lenguaje de programación compilado, de propósito general y multiparadigma que está siendo desarrollado por Mozilla. Ha sido diseñado para ser "un lenguaje seguro, concurrente y práctico. Es un lenguaje de programación multiparadigma que soporta programación funcional pura, por procedimientos, imperativa y orientada a objetos.

Según la política de Mozilla, Rust es desarrollado de forma totalmente abierta y busca la opinión y contribución de la comunidad. El diseño del lenguaje se ha ido perfeccionando a través de las experiencias en el desarrollo del motor de navegador Servo, y el propio compilador de Rust. Aunque es desarrollado y patrocinado por Mozilla y Samsung, es un proyecto comunitario. Una gran parte de las contribuciones proceden de los miembros de la comunidad.

Para el 2020 es uno de los lenguajes de programación más usados a la hora de trabajar con criptomonedas y crear nodos para minar cripto activos.

Descripción

El objetivo de Rust es ser un buen lenguaje para la creación de grandes programas del lado del cliente y del servidor que se ejecuten en Internet. Esto ha llevado a un conjunto de características con un énfasis en la seguridad, el control de distribución de la memoria y la concurrencia. Se espera que el rendimiento de código seguro sea más lento que C++, si el rendimiento es la única consideración, pero si lo comparamos con el código C++ hecho para que tome precauciones comparables a las que toma Rust, este último puede ser incluso más rápido.

La sintaxis de Rust es similar a la de C y C++, con bloques de código delimitados por llaves y estructuras de control de flujo tales como **if, else, do, while y for**. No todas las estructuras de C y C++ están presentes, además, otras (como la palabra clave match para ramificación multidireccional) serán menos familiares para programadores que vienen de estos lenguajes.

El sistema está diseñado para tener un acceso seguro a la memoria, y no permite punteros nulos o punteros colgantes. Los valores de los datos sólo se pueden inicializar a través de un conjunto fijo de formas, las cuales requieren que sus entradas hayan sido ya inicializadas.

El sistema de tipos soporta un mecanismo similar a las clases de tipos, llamado "traits", inspirados directamente por el lenguaje Haskell. Esta es una facilidad para el

polimorfismo que soporta distintos tipos de argumentos (polimorfismo ad-hoc), lograda mediante la adición de restricciones para escribir declaraciones de variables. Otras características de Haskell, como el polimorfismo de diferente tipo (higher-kinded), no están soportadas.

Ejemplos

El siguiente código es válido para Rust 0.8. En versiones posteriores puede cambiar la sintaxis o las funciones.

Programa que muestra la frase "¡Hola, mundo!":

```rust
fn main() {
    println!("¡Hola, mundo!");
}
```

Dos versiones de la función factorial, en el estilo recursivo e iterativo:

```rust
/* Las ramas en esta función exhiben los valores de retorno implícito opcional
   de Rust, que pueden ser usados cuando se prefiera un estilo más "funcional".
   A diferencia de C++ y otros lenguajes similares, la estructura de control
   `if` es una expressión en vez de una declaración, y por tanto tiene un valor
   de retorno propio. */

fn recursive_factorial(n: i32) -> i32 {
    if n == 0 {
        1
    } else {
        n * recursive_factorial(n-1)
    }
}

fn iterative_factorial(n: i32) -> i32 {
    // Las variables son declaradas con `let`.
    // La palabra `mut` permite que las variables puedan ser mutadas.
    let mut i = 1;
    let mut result = 1;
    while i <= n {
        result *= i;
        i += 1;
    }
    return result; // Un retorno explítico, en contraste con la función previa.
}

fn main() {
    println!("Resultado recursivo: {:i}", recursive_factorial(10));
    println!("Resultado iterativo: {:i}", iterative_factorial(10));
}
```

Una simple demostración de las capacidades de concurrencia ligera de Rust:

```
/* Esta función crea diez "tareas" que se pueden ejecutar concurrentemente.
   Ejecútalo múltiples veces y observa la salida irregular que se obtiene al
   estar cada tarea llamando al stdout, ya que cada tarea puede producirse entre
   las sucesivas llamadas a `println` y dentro de la función `println` en sí. */

fn main() {
    // Esta cadena es inmutable, para que pueda ser accedida de forma segura
    // por múltiples tareas.
    let message = "¡Miradme, soy un proceso ligero!";
    // Los bucles `for` funcionan con cualquier tipo que implemente el trait
    // `Iterator`.
    for num in range(0, 10) {
        do spawn {
            println(message);
            // `println!` es una macro que verifica estáticamente un string de
            // formato. Las macros son estructurales (como en Scheme) en lugar
            // de ser textuales (como en C).
            println!("Este mensaje ha sido ofrecido por la tarea {:i}.", num);
        }
    }
}
```

Interfaces gráficas (GUI)

Rust permite la creación de interfaces gráficas mediante las APIs nativas de la plataforma anfitriona, esto gracias a que tiene características de un lenguaje de bajo nivel. Sin embargo, esta ruta de desarrollo puede llegar a generar dificultades en proyectos que planten la admisión de múltiples plataformas.38

Actualmente existen múltiples desarrollos para crear GUIs en Rust, algunos permiten la creación de la interfaz junto a Electrón o haciendo uso de HTML, algunos otros, suministran bibliotecas nativas para el lenguaje, pero todavía se encuentran en un estado inmaduro de desarrollo. Finalmente están los proyectos que enlazan bibliotecas bastante conocidas y con una mayor madurez en su desarrollo, como GTK o Qt, lo cuales permiten desarrollos multiplataforma.39 Algunos proyectos populares son los siguientes:

Es una implementación de las bibliotecas de GTK 3 para el uso junto al lenguaje Rust. Para ello, crea funciones de Rust superpuestas a las funciones de las bibliotecas de GTK en C, haciendo uso de Foreign Function Interface que ofrece Rust.

Navegador web

Se están escribiendo en Rust un navegador web y varios componentes relacionados, que incluyen:
Firefox
Servo: motor de navegador web paralelo de Mozilla desarrollado en colaboración con Samsung
Quantum: un proyecto, compuesto por varios subproyectos, para mejorar el motor del navegador web Gecko de Firefox, desarrollado por Mozilla

Sistemas operativos

Muchos sistemas operativos (SO) y componentes relacionados se están escribiendo en Rust. A partir de enero de 2019, los sistemas operativos incluían: BlogOS, intermezzOS, QuiltOS, Redox, RustOS, Rux, Tefflin y Tock.
Redox: Un micronucleo
Stratis: Un sistema de archivos para Fedora y RHEL 8.

CYCLONE

El lenguaje de programación Cyclone está destinado a ser un dialecto seguro del lenguaje C. Cyclone está diseñado para evitar desbordamientos de búfer y otras vulnerabilidades que son posibles en los programas C, sin perder el poder y la conveniencia de C como herramienta para la programación del sistema.

El desarrollo de Cyclone se inició como un proyecto conjunto de AT&T Labs Research y el grupo de Greg Morrisett en Cornell en 2001. La versión 1.0 se lanzó el 8 de mayo de 2006.
Características del idioma
Cyclone intenta evitar algunos de los errores comunes de C, sin dejar de mantener su apariencia y rendimiento. Con este fin, Cyclone impone los siguientes límites a los programas:

- Se insertan verificaciones NULL para evitar fallas de segmentación
- La aritmética de punteros es limitada
- Los punteros deben inicializarse antes de su uso (esto se aplica mediante un análisis de asignación definido)
- Los punteros colgantes se evitan mediante el análisis de la región y los límites de free ()
- Solo se permiten moldes y uniones "seguras"
- ir a ámbitos no está permitido
- No se permiten las etiquetas de los interruptores en diferentes ámbitos.
- Las funciones de retorno de puntero deben ejecutar return
- setjmp y longjmp no son compatibles

Para mantener el conjunto de herramientas al que están acostumbrados los programadores de C, Cyclone proporciona las siguientes extensiones:

- Los punteros Never-NULL no requieren comprobaciones NULL
- Los punteros "gordos" admiten la aritmética de punteros con verificación de límites en tiempo de ejecución
- Las regiones de crecimiento admiten una forma de gestión de memoria manual segura
- Recolección de basura para valores asignados al montón
- Las uniones etiquetadas admiten argumentos que varían de tipo
- Las inyecciones ayudan a automatizar el uso de uniones etiquetadas para programadores
- El polimorfismo reemplaza algunos usos del vacío *
- los varargs se implementan como punteros gordos
- Las excepciones reemplazan algunos usos de setjmp y longjmp

- Para una mejor introducción de alto nivel a Cyclone, el razonamiento detrás de Cyclone y la fuente de estas listas, consulte este documento.

Cyclone se parece, en general, a C, pero debería verse como un lenguaje similar a C.
Tipos de puntero
Cyclone implementa tres tipos de puntero:

- (el tipo normal)
- @ (el puntero nunca NULL) y
- ? (el único tipo con aritmética de punteros permitida, punteros "gordos").

El propósito de presentar estos nuevos tipos de punteros es evitar problemas comunes al usar punteros. Tomemos, por ejemplo, una función, llamada foo que toma un puntero a un int:

```
int foo(int *);
```

Aunque la persona que escribió la función foo podría haber insertado comprobaciones NULL, supongamos que por razones de rendimiento no lo hizo. Llamando a foo (NULL); dará como resultado un comportamiento indefinido (normalmente, aunque no necesariamente, se envía una señal SIGSEGV a la aplicación). Para evitar tales problemas, Cyclone introduce el tipo de puntero @, que nunca puede ser NULL. Por lo tanto, la versión "segura" de foo sería:

```
int foo(int @);
```

Esto le dice al compilador Cyclone que el argumento de foo nunca debe ser NULL, evitando el comportamiento indefinido mencionado anteriormente. El simple cambio de * a @ evita que el programador tenga que escribir comprobaciones NULL y que el sistema operativo tenga que capturar las desreferencias del puntero NULL.

Este límite adicional, sin embargo, puede ser un gran obstáculo para la mayoría de los programadores de C, que están acostumbrados a poder manipular sus punteros directamente con aritmética. Aunque esto es deseable, puede provocar desbordamientos de búfer y otros errores de tipo "uno a uno". ¿Para evitar esto, el? El tipo de puntero está delimitado por un límite conocido, el tamaño de la matriz.

Aunque esto agrega sobrecarga debido a la información adicional almacenada sobre el puntero, mejora la seguridad y la protección. Tomemos, por ejemplo, una función strlen simple (e ingenua), escrita en C:

```c
int strlen(const char *s)
{
    int iter = 0;
    if (s == NULL)
       return 0;
    while (s[iter] != '\0') {
       iter++;
    }
    return iter;
}
```

Esta función asume que la cadena que se pasa termina con NULL ('\ 0'). Sin embargo, ¿qué pasaría si char buf [6] = {'h', 'e', 'l', 'l', 'o', '!'}; pasaron a esta cadena? Esto es perfectamente legal en C, pero haría que strlen iterara a través de la memoria no necesariamente asociada con la cadena s. Hay funciones, como strnlen, que se pueden utilizar para evitar tales problemas, pero estas funciones no son estándar en todas las implementaciones de ANSI C. La versión Cyclone de strlen no es tan diferente de la versión C:

```c
int strlen(const char ? s)
{
   int iter, n = s.size;
   if (s == NULL)
      return 0;
   for (iter = 0; iter < n; iter++, s++) {
      if (*s == '\0')
         return iter;
   }
   return n;
}
```

Aquí, strlen se limita a sí mismo por la longitud de la matriz que se le pasa, por lo que no supera la longitud real. ¿Cada uno de los tipos de puntero se puede convertir de forma segura a cada uno de los demás, y las matrices y cadenas se transmiten automáticamente a? por el compilador. (¿La conversión de? A * invoca una verificación de límites, y la conversión de? A @ invoca tanto una verificación NULL como una verificación de límites. La conversión de * a? No da como resultado ninguna verificación; ¿el puntero? Resultante tiene un tamaño de 1.)

Punteros colgantes y análisis de regiones

Considere el siguiente código, en C:

```c
char *itoa(int i)
{
   char buf[20];
   sprintf(buf,"%d",i);
   return buf;
}
```

La función itoa asigna una matriz de caracteres buf en la pila y devuelve un puntero al inicio de buf. Sin embargo, la memoria utilizada en la pila para buf se desasigna cuando la función regresa, por lo que el valor devuelto no se puede usar de forma segura fuera de la función. Si bien gcc y otros compiladores advertirán sobre dicho código, lo siguiente normalmente se compilará sin advertencias:

```c
char *itoa(int i)
{
   char buf[20], *z;
   sprintf(buf,"%d",i);
   z = buf;
   return z;
}
```

gcc puede producir advertencias para dicho código como un efecto secundario de la opción -O2 o -O3, pero no hay garantías de que se detecten todos esos errores. Cyclone realiza un análisis regional de cada segmento de código, evitando punteros colgantes, como el que se devuelve en esta versión de itoa. Todas las variables locales de un ámbito determinado se consideran parte de la misma región, separadas del montón o de cualquier otra región local. Por lo tanto, al analizar itoa, el compilador Cyclone vería que z es un puntero a la pila local y reportaría un error.

SIMILITUDES

El lenguaje natural es el que hablan las personas, mientras que el lenguaje de programación se destina a las máquinas. Ambos lenguajes presentan importantes similitudes, como la diferenciación que hacen entre sintaxis y semántica, su propósito de comunicar y la existencia de una composición base en los dos.

De forma general se observan varias semejanzas y puntos en común, pues los dos tipos fueron creados para comunicar ideas, expresiones e instrucciones, por ejemplo. Sin embargo, también es posible identificar algunas diferencias.

Cuando se trata de las diferencias entre el lenguaje natural y el lenguaje de programación, vale señalar que este último es más estricto y menos tolerante que el primero. Esto ocurre porque los lenguajes humanos tienen una redundancia incorporada que permite resolver alguna ambigüedad utilizando el contexto.

En cambio, los lenguajes de programación prácticamente no tienen redundancia, ya que de lo contrario sería muy fácil caer en ambigüedad y no indicar el comando correcto.

El hecho de que los lenguajes de programación sean más estrictos en este aspecto se debe a que las computadoras son muy precisas en las instrucciones que les gusta recibir. Además, las máquinas no cuentan con la capacidad de aclarar el significado de una expresión como lo haría un ser humano.

Asimismo, resulta indispensable que los lenguajes de programación sean fijos y cerrados para evitar confusiones, mientras que los lenguajes naturales son abiertos y admiten combinaciones sin caer en el riesgo de equivocaciones.

Debido a estas circunstancias, los lenguajes de programación están diseñados desde cero por desarrolladores talentosos enfocados en no arruinar las distintas funcionalidades, ya sea el control de un robot, los equipos médicos, las máquinas espaciales o un juego de Facebook, por ejemplo.

Principales características de los lenguajes de programación

La popularidad de un lenguaje de programación depende de las funcionalidades y utilidades que proporcione a los programadores. Las características que debe tener un lenguaje de programación para destacar son las siguientes:

- Simplicidad: el lenguaje debe ofrecer conceptos claros y simples que faciliten su aprendizaje y aplicación, de manera que sea sencillo de comprender y mantener. La simplicidad no significa que se le pueda restar el poder óptimo de funcionamiento.
- Naturalidad: se refiere a que su aplicación en el área para la que fue diseñado debe hacerse de forma natural, proporcionando operadores, estructuras y sintaxis para que los operadores trabajen eficientemente.
- Abstracción: consiste en la capacidad de definir y utilizar estructuras u operaciones complicadas ignorando algunos detalles. Esto influye en la capacidad de escritura.
- Eficiencia: los lenguajes de programación deben traducirse y ejecutarse eficientemente para no ocupar demasiado espacio en la memoria ni gastar mucho tiempo.
- Estructuración: permite que los programadores escriban sus códigos de acuerdo con los conceptos de programación estructurada, para evitar la creación de errores.
- Compacidad: con esta característica es posible expresar las operaciones con concisión, sin tener que escribir demasiados detalles.
- Localidad: se refiere a que los códigos se concentran en la parte del programa con la cual se está trabajando en un momento determinado.

¿Cuál es el mejor lenguaje de programación web?

El desarrollo web puede hacerse a través de diferentes lenguajes de programación que permiten construir un sitio o diseñar una aplicación. Definir cuál es el mejor lenguaje de programación web es complicado porque depende de varios factores. No obstante, destacan las siguientes opciones:

- Java: lenguaje multiuso que se ajusta eficientemente al desarrollo web.
- Go: es de propósito general y ofrece un lenguaje flexible que facilita la creación de aplicaciones.
- Ruby on Rails: permite diseñar aplicaciones web rápidamente.
- Python: funciona en una amplia variedad de contextos y en la web tiene ventajas técnicas.
- JavaScript: se ubica del lado del cliente y se puede extender al servidor para distintas funciones.

LENGUAJE	¿QUE ES?	VENTAJAS	DESVENTAJAS	SISTEMA OPERATIVO
C++	Lenguaje de programación orientada a objetos creado por Bjarne Stroustrup	Es potente en cuanto a lo que se refiere a creación de sistemas complejos un lenguaje muy robusto	No es atractivo visualmente	Sirve para todos los sistemas operativos pero cada uno con su respectiva versión para dicho sistema
			No soporta para creación de páginas web	
HTML	Lenguaje utilizado para la creación de páginas web. HTML significa "hypertext mark-up language", que significa, "lenguaje para el formato de documentos de hipertexto", es decir, texto presentado de forma estructurada y agradable, con enlaces (hyperlinks) que conducen a otros documentos o fuentes	Sencillo que permite describir hipertexto.	Lenguaje estático.	Se pueden ver en todos los sistemas operativos, pero la interpretación varía según el navegador que esté utilizando.
		Archivos pequeños.	La interpretación de cada navegador puede ser diferente.	
		Lenguaje de fácil aprendizaje.	Guarda muchas etiquetas que pueden convertirse en "basura" y dificultan la corrección.	
		Lo admiten todos los exploradores	Lento para ejecutar archivos pesados	
PHP	PHP es un lenguaje de programación interpretado, diseñado originalmente para la creación de páginas web dinámicas	Es un lenguaje multiplataforma.	Como es un lenguaje que se interpreta en ejecución, para ciertos usos puede resultar un inconveniente que el código fuente no pueda ser ocultado. La ofuscación es una técnica que puede dificultar la lectura del código pero no la impide y, en ciertos casos, representa un costo en tiempos de ejecución.	Se usa principalmente para la interpretación del lado del servidor, páginas web y CMS
		Orientado al desarrollo de aplicaciones web		Se usa en todos los sistemas operativos

VISUAL BASIC	Es un lenguaje de programación dirigido por eventos, desarrollado por el alemán Alan Cooper para Microsoft.	Posee una curva de aprendizaje muy rápida.	Las críticas hechas en las ediciones de visual Basic anteriores a vb.net son variadas, se citan entre ellas: Problema de versionado asociado con varias librerías DLL, conocido como DLL HELL.	Sirve para hacer aplicaciones de escritorio
		Integra el diseño e implementación de formularios de Windows.	Pobre soporte para programación orientada a objetos	
		Permite usar con facilidad la plataforma de los sistemas Windows, dado que tiene acceso prácticamente total al api de Windows, incluidas librerías actuales.	Incapacidad para crear aplicaciones multihilo, sin tener que recurrir a llamadas del api de Windows.	
		Es uno de los lenguajes de uso más extendido, por lo que resulta fácil encontrar información, documentación y fuentes para los proyectos.		
		Fácilmente extensible mediante librerías DLL y componentes ActiveX de otros		
C#	Es un lenguaje de programación orientado.	Declaraciones en el espacio de nombres: al empezar a programar algo, se puede definir una o más clases dentro de un mismo espacio de nombres.	Se tiene que conseguir una versión reciente de visual studio .net, por otra parte se tiene que tener algunos requerimientos mínimos del sistema para poder trabajar adecuadamente tales como contar con Windows nt 4 o superior, tener alrededor de 4 gigas de espacio libre para la pura instalación, etc.	La plataforma .Net Sirve para hacer aplicaciones de escritorio, aplicaciones web y móviles.
	A objetos desarrollado y estandarizado por Microsoft como parte de su plataforma net.	Tipos de datos: en c# existe un rango más amplio y definido de tipos de datos que los que se encuentran en c, c++ o java.		Sistema operativo Windows
	Los programadores le consideran el primo hermano de JAVA	Atributos: cada miembro de una clase tiene un atributo de acceso del tipo público, protegido, interno, interno protegido y privado.		

JAVA	Es un lenguaje orientado a objetos, de una plataforma independiente, fue desarrollado por la compañía SUN Microsystems ahora es propietario ORACLE.	Se pueden realizar distintos aplicativos, como son applets, que son aplicaciones especiales, que se ejecutan dentro de un navegador al ser cargada una página HTML en un servidor web, por lo general los applets (Ya son historia) son programas pequeños y de propósitos específicos.	Esperar la actualización siguiente para que sea más rápido.	Sirve para todos los sistemas operativos y si no es la versión adecuada para dicho sistema, la misma aplicación java se encarga de descargas o actualizar versión para un excelente desempeño en el pc.
	Maneja algunas plataformas de desarrollo:	Puede desarrollar aplicaciones de escritorio que se ejecutan en forma independiente, es decir con la programación java, se pueden realizar aplicaciones como un procesador de palabras, una hoja que sirva para cálculos, una aplicación gráfica, etc.		Algunos de los sistemas operativos más destacados en los que funciona la aplicación:
	Java Platform, Standard Edition o Java SE	Se puede realizar soluciones empresariales en un		**Unix, Linux, Solaris,**
	Java Platform Enterprise Edition o Java EE	Soporta el desarrollo de aplicaciones móviles		**Windows, mac.**
	Java Platform Micro Edition o Java ME			

TIPOS DE MEMORIA

¿Qué es la memoria?

La *memoria* es uno de los componentes fundamentales para el correcto funcionamiento de nuestra PC, ya que su existencia permite que la computadora pueda arrancar, se procesen los datos, se ejecuten las instrucciones para los distintos programas y demás.
Por otro lado, **cuanto mayor es la cantidad de memoria que posea una PC, mayor será el rendimiento** y la mejora en la performance del equipo.

No obstante, una computadora trabaja con cuatro tipos de memorias diferentes, que sirven para realizar diversas funciones. Estas son **la memoria RAM, la memoria ROM, la memoria SRAM o Caché y la memoria Virtual o de Swap.**

Tipos de memoria:

La memoria RAM

La más importante es la denominada memoria **RAM (Random Access Memory),** ya que nuestra computadora no podría funcionar sin su existencia.
En la RAM se guarda distinto tipo de información, desde los procesos temporales como modificaciones de archivos, hasta las instrucciones que posibilitan la ejecución de las aplicaciones que tenemos instaladas en nuestra PC.

Por tal motivo, es utilizada constantemente por el microprocesador, que accede a ella para buscar o guardar temporalmente información referente a los procesos que se realizan en la computadora.
Dentro de las memorias RAM existen distintos tipos de tecnologías que se diferencian principalmente por su velocidad de acceso y su forma física. Entre ellas encontramos las DRAM, SDRAM, RDRAM, entre otras.
Las denominadas DRAM (Dynamyc Random Acces Memory) han sido utilizadas en las computadoras desde los primeros años de la década de los 80's, y aún en la actualidad continúan utilizándose. Se trata de uno de los tipos de memorias más económicas, aunque su mayor desventaja está relacionada con la velocidad de proceso, ya que es una de las más lentas, lo que ha llevado a los fabricantes a modificar su tecnología para ofrecer un producto mejor.
En cuanto al tipo de tecnología SDRAM, derivada de la primera, comenzó a comercializarse a finales de la década de los 90's, y gracias a este tipo de memoria se lograron agilizar notablemente los procesos, ya que puede funcionar a la misma velocidad que la motherboard a la que se encuentra incorporada.
Por su parte, la tecnología RDRAM es una de las más costosas debido a su complejidad de fabricación, y sólo se utilizan en procesadores grandes, tales como los Pentim IV y superiores.
Otra de las diferencias entre las distintas memorias RAM se halla en el tipo de módulo del que se trate, que pueden ser SIMM (Single in line Memory Module), DIMM (Double Memory Module) y RIMM (Rambus in line Memory Module), dependiendo de la cantidad de pines que contenga y del tamaño físico del módulo.

La memoria ROM

Además de la memoria RAM, las computadoras trabajan con la memoria denominada ROM, Read Only Memory, que como su nombre lo indica se trata de una memoria sólo de lectura, ya que la mayoría de estas memorias no pueden ser modificadas debido a que no permiten su escritura.
La memoria ROM viene incorporada a la motherboard y es utilizada por la PC para dar inicio a la BIOS, lo cual es básicamente un programa que posee las instrucciones adecuadas para guiar a la computadora durante el arranque.

Entre sus funciones, la BIOS comienza con el proceso denominado POST (Power On Self Test) durante el cual inspeccionará todo el sistema para corroborar que todos sus componentes funcionan adecuadamente para dar lugar al arranque.
Para ello, la BIOS consulta un registro en el que se halla toda la información referente al hardware que tenemos instalado en nuestra PC, para comprobar que todo se encuentre en orden. Dicho registro es denominado CMOS Setup.
Si bien mencionamos que en muchos casos la memoria ROM no puede ser modificada, en la actualidad gran cantidad de motherboards incorporan nuevos modelos de ROM que permiten su escritura, para que el usuario pueda realizar cambios en la BIOS con el fin de mejorar su funcionamiento.
La diferencia fundamental que existe entre la memoria RAM y la ROM radica en la velocidad, ya que la ROM al tratarse de un tipo de memorial secuencial necesita recorrer todos los datos hasta hallar la información que está buscando, mientras que la RAM trabaja de manera aleatoria, lo que hace que acceda a la información específica de manera directa.
Este factor hace que la velocidad de la RAM sea notablemente superior. Asimismo, la capacidad de ésta es mayor a la de la memoria ROM, y a diferencia de esta última, la RAM no viene integrada al motherboard, lo que permite que el usuario pueda expandir la cantidad de memoria RAM de su PC.

La memoria caché

Otro de los tipos de memoria utilizados por las computadoras es la denominada SRAM, más conocida como memoria Caché.
Tanto el procesador como el disco rígido y la motherboard poseen su propia memoria caché, que básicamente resguarda distintas direcciones que son utilizadas por la memoria RAM para realizar diferentes funciones, tales como ejecutar programas instalados en la PC.
El proceso que realiza la memoria caché es guardar las ubicaciones en el disco que ocupan los programas que han sido ejecutados, para que cuando vuelvan a ser iniciados el acceso a la aplicación logre ser más rápido.

Existen tres tipos de caché diferentes:

- El caché L1 que se encuentra en el interior del procesador y funciona a la misma velocidad que éste, y en el cual se guardan instrucciones y datos.
- El caché L2 que suelen ser de dos tipos: interno y externo. El primero se encuentra dentro de la motherboard, mientras que el segundo se halla en el procesador, pero de manera externa, lo que lo hace más lento que el caché L1.
- El caché L3 que sólo vienen incorporado a algunos de los microprocesadores más avanzados, lo que resulta en una mayor velocidad de procesos.

La memoria de Swap

En algunas computadoras, sobre todo en aquellas que poseen sistema operativo Microsoft Windows o Linux, también encontraremos la denominada memoria virtual o de Swap.

Este tipo de memoria, que funciona de manera similar a la caché, es creado por Windows o Linux para ser utilizada exclusivamente por el sistema operativo. En el caso de Linux esta denominada memoria swap generalmente está ubicada en una partición diferente del disco, mientras que en el sistema de Microsoft es un archivo dentro del sistema operativo mismo.

En muchas ocasiones la memoria virtual suele producir ciertos problemas que ocasionan que la PC se cuelgue, ya que este tipo de memoria ha sido creada por el sistema dentro del disco rígido y a veces puede llegar a superar la capacidad de proceso.
En la ejecución de programas mediante la memoria virtual, sólo obtendremos como resultado que nuestra PC se vuelva más lenta, ya que le resta velocidad de proceso al disco rígido.

La mejor forma de evitar este inconveniente es expandir la cantidad de memoria RAM de nuestra PC, para que el sistema no necesite de la creación de memoria virtual extra, y por ende relentice los procesos durante nuestro trabajo.

Tipos de memorias RAM

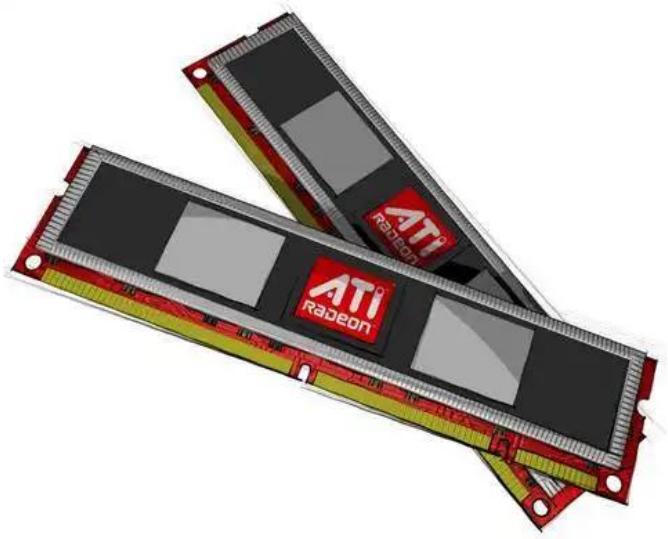
Memorias DDR

De acuerdo al tipo de placa madre que utilicemos en nuestra PC, ésta estará provista de diferentes tipos de zócalos según su antigüedad, y puede que utilice memoria RAM DDR, DDR2, DDR3 o DDR4.
Las siglas DDR son utilizadas para abreviar el concepto "Double Data Rate", cuya definición es memoria de doble tasa de transferencia, y se trata de una serie de módulos que están compuestos por memorias síncronas, llamadas SDRAM, y si bien tienen el mismo tamaño de los DIMM de SDRAM, las DDR-SDRAM poseen mayor cantidad de conectores, ya que mientras la SDRAM normal tiene 168 pines, la DDR-SDRAM posee 184.

Las memorias DDR trabajan transfiriendo datos a través de dos canales diferentes, de manera simultánea y en un mismo ciclo de reloj con una transferencia de un volumen de información de 8 bytes en cada ciclo de reloj. No obstante, son compatibles con procesadores más potentes en cuanto a ciclos de reloj.
En lo que respecta a la memoria DDR2 se trata básicamente de la segunda generación de DDR SDRAM, que ha logrado mejorar ciertos aspectos brindando mayor rapidez en los procesos simultáneos.
Al ser una tecnología más moderna, las DDR2 poseen notables diferencias con sus antecesoras, entre las cuales la más significativa tiene que ver con el valor de transferencia mínima, ya que mientras que en las DDR tradicionales es de 1600Mbps, en las DDR2 se duplica a 3200Mbps.

Esto le permite un mayor ancho de banda en los procesos, ya que las memorias DDR2 tienen mayor latencia porque trabajan con 4 bits por ciclo (2 de ida y 2 de vuelta) dentro de un mismo ciclo y bajo la misma frecuencia de una DDR convencional.
Lamentablemente las DDR y las DDR2 no son compatibles, por lo que si tienes una PC cuya motherboard posee zócalos para DDR no podrás utilizar Memorias DDR2, ya que estás últimas tienen 240 pines, lo que permite reducir su voltaje a 1.8V, mientras que las DDR utilizan un voltaje de 2.5V.
La reducción del voltaje en la segunda generación de memorias DDR han incorporado una gran mejora, debido a que de esta manera se reduce considerablemente el consumo de energía y por ende la generación de calor.
El avance en el desarrollo de la tecnología de este tipo de memorias RAM produjo los módulos DDR3, cuyo fabricante más importante hasta el momento ha sido la empresa Samsung Electronics.
DDR3 incorpora importantes mejoras en el campo de las memorias DDR SDRAM, entre las que se destaca el hecho de que puede transferir datos a una tasa de reloj efectiva de 800-1600 Mhz, superando en gran medida a las DDR anteriores, ya que las DDR2 tienen una tasa de 533-800 MHz y las DDR de 200-400 MHz.

Esto permite un mayor ancho de banda en los procesos, significativamente notable en el funcionamiento de la PC, además de haber duplicado su latencia a 8 bits, con el fin de aumentar su rendimiento, y duplicar su tasa de transferencia mínima a 6400Mbps, en comparación a las DDR2 que poseen una tasa de 3200Mbps.
Las DDR3 consumen sólo 1.5V, gracias a la implementación de la tecnología de fabricación de 80 nanómetros. Este cambio reduce el consumo de energía y la generación de calor, por lo que aumenta la velocidad en los procesos.
En cuanto al aspecto físico, si bien las DDR3 poseen 240 pines, es decir la misma cantidad que las DDR2, ambos tipos de memorias son incompatibles, ya que los pines han sido ubicados de manera diferente.



Las memorias DDR4 poseen una velocidad de 2.667 Mhz y su tasa de transferencia es de 21.300 Mbps.

Memorias GDDR

En el mercado, además de las típicas memorias RAM del tipo DDR, también podemos encontrarnos con una variante de la misma, llamada GDDR SDRAM (Graphics Double Data Rate Synchronous Dynamic RAM), la cual es un tipo de memoria que fue diseñada específicamente con el propósito de ser utilizada en el ámbito del renderizado de vídeo, habitualmente trabajando en equipo con la GPU de nuestra tarjeta gráfica.

Con este tipo de memorias, estaremos en condiciones de crear estructuras gráficas 3D muy complejas, para las cuales necesitamos gran cantidad de memoria. Sin embargo, con las memorias GDDR, que son mucho más rápidas, la cantidad de memoria requerida para estos procesos se reduce, lo que significa menos dinero y espacio, aunque el precio de las memorias GDDR no permite que puedan ser usadas por el usuario promedio con un presupuesto ajustado en sus implementaciones domésticas, ya que son mucho más caras de producir que las DDR, lo que se traduce en un precio mucho mayor.

Si bien las memorias de tipo GDDR comparte muchas de las características técnicas con las memorias de tipo DDR, lo cierto es que no son completamente iguales. En este sentido, las memorias GDDR, al estar optimizadas para su uso en el renderizado de video, prioriza el ancho de banda, no a la latencia. También las memorias GDDR trabajan respetando el estándar DDR especificado por la JEDEC, por lo cual es capaz de enviar dos bits o 4 por cada ciclo de reloj, si bien en este caso la memoria GDDR está optimizada para lograr frecuencias mayores y un ancho de bus más grande, lo que le permite minimizar el tiempo de acceso a las instrucciones almacenadas en la memoria.

Las memorias GDDR, al igual que las DDR, con el tiempo fueron evolucionando, por lo cual podemos encontrar múltiples variantes. A partir de este punto conoceremos los **diferentes tipos de memorias GDDR** en el mercado.

GDDR: El primer tipo de GDDR en el mercado. Su frecuencia efectiva de trabajo era de entre 166 y 950 MHz con una latencia de 4 a 6 ns.

GDDR2: En este tipo se mejoró la frecuencia de operación, que llegó a oscilar entre los 533 y 1000 MHz, y podían ofrecer un ancho de banda de entre 8,5 a 16 GB/s.

GDDR3: Especialmente utilizadas por algunos modelos de tarjetas gráficas de ATI y Nvidia, estas memorias pueden operar entre los 166 y 800 MHz.

GDDR4: Reemplazadas rápidamente por las GDDR5, sólo fueron utilizadas por algunos modelos de AMD.

GDDR5: Un tipo de memoria GDDR de los más extendidos en los últimos años. Es utilizada en tarjetas de video de gama media y alta de fabricantes como Nvidia, AMD y Radeon, entre otros. Estas memorias son capaces de **ofrecer un ancho de bus cercanos a los 20 GB/s** en buses de 32 bits y a los 160 GB/s en buses de 256 bit, pudiendo llegar la frecuencia de operación hasta los 8 Gbps. Cabe destacar que este tipo de memorias se instalan también en consolas de juegos como la Xbox One y la PS4.

GDDR5X: Esta memoria es básicamente una **evolución de la tecnología GDDR5** que es utilizada en algunos modelos de tarjetas de video. Ofrecen una frecuencia de operación de 11 Gbps y un ancho de banda de 484 GB/s sobre un bus de 352 bit.

GDDR6: Hasta el momento, es la **última versión de memoria GDDR disponible.** Son capaces de ofrecer hasta una frecuencia de operación de 14 Gbps con un ancho de banda de 672 GB/s sobre un bus de 384 bit. Este tipo de memorias se utilizan en tarjetas de video de alta gama como las Nvidia Titan RX.

Anexo 1: La memoria ROM

La memoria ROM es quizás el elemento de hardware más importante de computadoras y dispositivos portátiles como celulares, teléfonos inteligentes y tablets, entre muchos otros, ya que en este pequeño componente electrónico se almacena toda la información necesaria para que el dispositivo arranque y pueda cumplir con su función.

El término ROM es una abreviatura del término sajón “Read Only Memory” que en español significa “Memoria de solo lectura”, y como su nombre lo indica, este tipo de memoria almacena información a la cual sólo puede ser accedida, es decir no puede escribirse con nuevos datos, salvo mediante procedimientos especiales como cuando estamos actualizando una BIOS.

¿Qué es la memoria ROM?

Básicamente, una memoria ROM es un chip que en su interior almacena la información necesaria para poder arrancar un dispositivo electrónico como una computadora o un smartphone, y cuya principal característica es la de tener la capacidad de conservar los datos que contiene aun cuando no existan energía que la alimente, al contrario que las memorias RAM, las cuales, si no son energizadas, pierden inmediatamente su contenido. El término ROM en la actualidad se utiliza por convención, y provienen básicamente de cuando las memorias ROM se desarrollaban y salían de la factoría ya con los datos almacenados en ellas, y no existía ninguna forma de poder escribirlas.

Hoy en día es posible encontrar memorias que cumplen con la misma función de las antiguas ROM pero que sí se pueden escribir, llamadas EPROM y Flash EEPROM, sin embargo, escribir en este tipo de memorias es una tarea complicada y que no se puede hacer directamente, salvo con herramientas y procedimientos especiales, que la mayoría de las veces no están al alcance del usuario promedio.
Estas memorias EPROM y Flash EEPROM pueden escribirse multitud de veces, lo que favorece, por ejemplo, que actualizar la BIOS de una computadora pueda ser una tarea frecuente y que no presente problemas. Tal es la adopción de este tipo de memorias para cumplir con el rol de ROM que prácticamente no podremos encontrar en el mercado dispositivos que contenga ROM del tipo más antiguo desde finales de la primera década del siglo XXI.

¿Para qué sirve la memoria ROM?

Las memorias ROM en los dispositivos cumplen con la importante función de almacenar en su interior el código que se necesita para arrancar los diferentes módulos que componen una computadora, es decir todo lo que se requiere para comenzar a trabajar con ella. Asimismo, la memoria ROM cumple con la función de iniciar el sistema operativo de la PC en que se encuentra instalado.
Además de utilizarse para la gestión del proceso de arranque de la PC, la memoria ROM se usa para el chequeo inicial del sistema y diversas rutinas de control de dispositivos de entrada y salida.

La capacidad que ofrece la memoria ROM de poder conservar los datos, aunque no se encuentre energizada, la hace ideal para el trabajo de iniciar una computadora, ya que los datos almacenados en la memoria ROM no se alterna ni degradan en ausencia de electricidad que la alimente, es decir siempre son los mismos, por lo cual el dispositivo que gestionan siempre se comportará de la misma manera.

Tipos de memoria ROM

Con el paso de los años, las memorias ROM han ido evolucionando para adaptarse a las nuevas tecnologías. En la actualidad, **existen tres tipos básicos de memoria ROM.**

ROM (Read Only Memory)

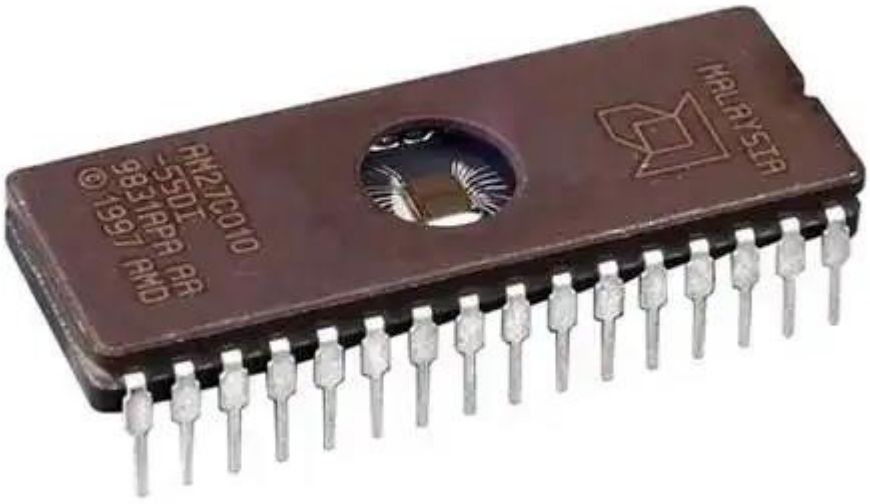
Este tipo de memoria ROM o “Memoria de solo lectura” fue la primera que se desarrolló y fabricó, y la información que debía almacenarse en ella se grababa usando un procedimiento que implicaba la utilización de una placa de silicona y una máscara. Este tipo de memorias ROM ya no se utilizan, siendo reemplazadas por las memorias que se detallan a continuación.

PROM (Programmable Read Only Memory)

Las memorias PROM, también conocidas como “Memoria Programable de Sólo Lectura”, vieron la luz a fines de los 70s, y su programación, es decir la carga de los datos que debían contener, se efectuaba quemando unos determinados componentes electrónicos, llamados diodos, con una sobrecarga de tensión mediante un dispositivo conocido como “Programador ROM”. Los diodos afectados con la carga corresponden a “0”, mientras que los demás corresponden a “1”.

EPROM (Erasable Programmable Read Only Memory)

Las memorias del tipo EPROM, también conocidas como “Memoria Programable y Borrable de Sólo Lectura”, son básicamente memorias del tipo PROM pero que tienen la particularidad de poder borrarse. El modo de programar estas memorias es a través de rayos de luz ultravioleta que penetran en el circuito a través de una ventana en el encapsulado del chip. En el momento en que el chip se somete a la luz ultravioleta, todos los bits vuelven a su estado ”1”.



EEPROM (Electrically Erasable Programmable Read Only Memory)

Las memorias EEPROM conocidas también por el nombre “Memoria Programable de Sólo Lectura Borrable Eléctricamente”, son, al igual que las memorias PROM, borrables, sin embargo, estos procedimientos en las memorias EEPROM es más sencillo, ya que se puede realizar mediante una determinada corriente eléctrica.

Cabe destacar que las memorias EEPROM ofrecen una variante llamada Flash EEPROM, que utiliza menos componentes, específicamente un solo transistor, en lugar de los 2 ó 3 que utiliza la memoria EPROM. Además, ofrece la posibilidad de leer registro por registro, en vez de una lectura de página completa como la memoria EEPROM**.**

Diferencias entre memorias RAM y ROM

Como sabemos, existen dos tipos de memoria en una computadora, la memoria ROM y la memoria RAM, y cada una de ellas cumple con una función muy distinta. La memoria RAM, o memoria de acceso aleatorio, es aquella memoria a la que accede el sistema operático para buscar los datos que están usando tanto el usuario como el sistema operativo, ya que es un método mucho más rápido que buscarlos en el disco rígido.
La memoria RAM se puede leer y escribir múltiples veces, sin embargo, la RAM es temporal, ya que los datos que contiene se borran inmediatamente ante la falta de energía, es decir cuando pierde el suministro eléctrico.

En cambio, la memoria ROM no es afectada por el suministro eléctrico, lo que convierte a este tipo de memoria en el medio ideal para almacenar los datos necesarios para que un dispositivo funcione. Además, la condición de no ser escribible, por lo menos por los medios habituales que tiene disponible el usuario promedio, garantiza que mantendrá los datos que contiene en cualquier situación, por lo cual el dispositivo siempre encenderá y seguirá la misma rutina.

La memoria Caché

La memoria cache nació cuando se descubrió que las memorias ya no eran capaces de acompañar a la velocidad del procesador, haciendo que muchas veces este último se quedara "esperando" por los datos que debía entregar la memoria RAM para poder concluir sus tareas, perdiendo mucho rendimiento.
Si en la época del 386, año 1991, la velocidad de las memorias ya era un factor limitante, imagina este problema hoy, con los procesadores que tenemos actualmente.

Para solucionar este problema, se comenzó a usar la memoria cache, un tipo ultrarrápido de memoria que sirve para almacenar los datos que son más frecuentemente utilizados por el procesador, evitando, la mayoría de las veces, tener que recurrir a la comparativamente lenta memoria RAM.
Sin la memoria cache, la performance del sistema estaría limitada a la velocidad de la memoria, pudiendo caer hasta un 95%!

Los tipos de memoria cache

Se utilizan dos tipos de memoria cache, llamados cache primario, o *cache L1* (level 1), y cache secundario, o *cache L2* (level 2). La memoria cache primaria está insertada en el mismo procesador y es tan rápida como para acompañarlo en velocidad. Siempre que un nuevo procesador es desarrollado, es preciso desarrollar también un tipo más rápido de memoria cache para acompañarlo. Como este tipo de memoria es extremadamente cara (llega a ser centenares de veces más cara que la memoria RAM convencional) se usa sólo una pequeña cantidad de ella. Para complementar, se utiliza también un tipo de memoria cache un poco más lenta, la cual se llama cache secundario, que, por ser mucho más barata, permite usar mayor cantidad.

¿Como instalar memoria cache?

Primero, debes asegurarte que la placa madre permita la instalación de memoria cache. Las placas madre que permiten la instalación, poseen un socket llamado COAST donde se coloca el módulo de memoria cache. Generalmente se necesita cambiar los jumpers de configuración del tamaño de la memoria cache. La posición correcta de los jumpers se deberá consultar en el manual de la placa. Si luego de esta configuración la PC no enciende, significa que el módulo de memoria cache está fallado o es incompatible con

la placa madre. En este caso, el módulo debe ser cambiado. Cuando esté todo funcionando, se deberá habilitar el cache de memoria en la BIOS de la PC.

¿Chequear la existencia de memoria cache en la PC?

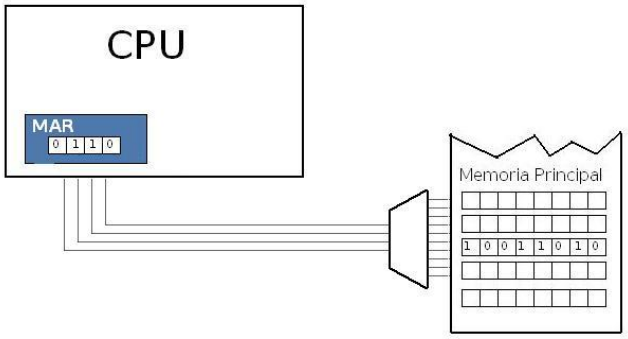
Hay varios programas para este fin. Uno de ellos se llama PC-Config, es shareware y puede ser bajado gratis en internet. Además de probar el cache, este programa no brinda información importante sobre la PC, tales como el tipo de memoria instalada y el tipo de chipset.

REGISTROS DE MEMORIA

Registro MAR

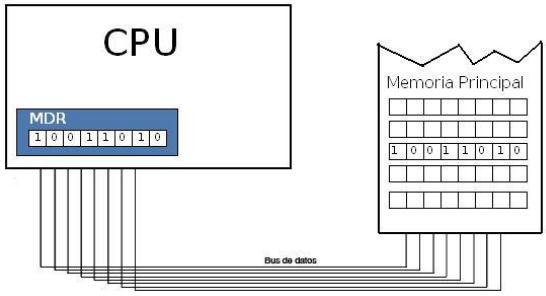
En arquitectura de ordenadores, el Memory Address Register (MAR), en español Registro de Direcciones de Memoria, es un registro específico de alta velocidad, integrado en el microprocesador. Este registro contiene la dirección del dato que se quiere leer o escribir. El registro está conectado con el bus de direcciones, y su contenido se refleja en este bus.

El número de direcciones que se pueden direccionar con una CPU depende del tamaño del MAR. Si el MAR tiene n bits de tamaño entonces se podrán direccionar un máximo de 2n palabras.



Registro MDR

En arquitectura de ordenadores, Memory Data Register (MDR), en español Registro de Datos de Memoria, es un registro específico de alta velocidad y poca capacidad integrado en el microprocesador. El registro está conectado al bus de datos y a través de él, el CPU lee o escribe un dato a dicho bus, que a continuación llegará a la memoria o a un puerto de entrada/salida.



Registro de pila

Un registro de pila es un registro de una CPU de computadora cuyo propósito es mantener la pista de la posición actual de la pila de llamadas. En una máquina de arquitectura basada en acumulador, este puede ser un registro dedicado como el puntero de pila (SP del inglés stack pointer) de una máquina Intel x86. En una máquina de registro general, puede ser un registro reservado por convención, como el de las máquinas PDP-11 o RISC. Algunos diseños como el Data General Eclipse no tenían ningún registro dedicado para el puntero de pila, pero usaron una dirección de memoria de hardware reservada para esta función.

Antes de finales de los años 1960, las máquinas como el PDP-8 y el HP 2100 no tuvieron compiladores que soportaran la recursión. Sus instrucciones de subrutinas, típicamente guardarían la localización actual de la dirección del salto, y después fijarían el contador de programa a la dirección siguiente.1 Mientras que esto era más simple que mantener una pila, ya que solamente hay una localización de retorno por sección de código de subrutina, de esta manera no podía haber recursión sin un esfuerzo considerable por parte del programador.

A diferencia de una máquina de registro, una máquina de pila tiene dos o más pilas. Una máquina de dos pilas (similar a la usada en la implementación del lenguaje Forth), tiene una pila de llamadas, que mantiene información sobre las llamadas a subrutinas en ejecución (que no han retornado) entre otras cosas, y el otro es una pila de parámetros, que mantiene información sobre los parámetros o datos con los que trabajan las subrutinas.

Registro índice

Afecta a los operandos durante la ejecución de un programa de computadora. El registro índice es típicamente usado para hacer operaciones de vectores/arreglos. Los registros de índice fueron usados por primera vez en 1949 en la computadora británica Manchester Mark I.

Los registros de índice dan formato al acceso de la memoria por accesos aleatorios. Son usados para una clase especial de direccionamiento indirecto donde una constante inmediata, es decir, que es parte de la instrucción en sí misma, es agregada al contenido de un registro para formar la dirección del operando o los datos reales; las arquitecturas que permiten que más de un registro sea usado de esta manera tienen un campo de opcode para especificar qué registro usar.

En las primeras computadoras, sin ninguna forma de direccionamiento indirecto, las operaciones con arreglos tenían que ser realizadas. o repitiendo linealmente el código

del programa para cada elemento del arreglo (es decir sobre todas las localizaciones de dirección), o usando técnicas de código automodificante bastante "sucias". Ambas alternativas conducían a desventajas absolutamente significativas en flexibilidad y mantenimiento del programa, así como a ser derrochadoras de la memoria del computador; el último método fue un recurso muy raro/escaso?? en instalaciones de computadora de la era inicial (así como en los primeros microcomputadores de varias décadas más tarde).

En general, los registros índices se convirtieron en una parte estándar de los computadores durante la segunda generación de la tecnología (más o menos por 1955-1964). Vea, por ejemplo, el mainframe IBM 700/7000. Las primeras "pequeñas máquinas" con registros índice incluyen al AN/USQ-17, alrededor de 1960, y los computadores en tiempo real (real time) de Scientific Data Systems. El primer microprocesador con un registro índice parece haber sido el Motorola 6800, cuyo clon mejorado, el MOS Technology 6502, hizo buen uso de dos registros índice.

PROPORNER UNA SOLUCION ALTERNA

Obtener ayuda con los errores de actualización e instalación de Windows 10

Existen muchos motivos por los que podrías recibir un mensaje de error durante la actualización o instalación de Windows 10, pero puedes corregir los errores más comunes siguiendo por ti mismo los pasos que se describen a continuación. Nota: una actualización lleva tu dispositivo desde una versión anterior de Windows, como Windows 7 o Windows 8.1, a Windows 10.

Antes de buscar un código de error específico, prueba las sugerencias enumeradas en Correcciones de errores en general. Si con estas no se soluciona tu problema de actualización o instalación, consulta la tabla de códigos de error de la parte inferior de este artículo.

Correcciones de errores en general
Estas son algunas cosas que puedes probar para corregir los errores de instalación y actualización:

- Quitar hardware externo
- Actualizar Windows
- Desinstala cualquier software antivirus que no sea de Microsoft.
- Desinstalar el software que no sea indispensable
- Liberar espacio en disco
- Errores 0xC1900101

Los errores que empiezan por 0xC1900101 suelen ser errores de controlador. Si ves alguno de estos códigos de error, intenta seguir los pasos que se describen a continuación para solucionar el problema. Si estos pasos no funcionan, consulte resolver errores de actualización de Windows 10 para obtener información técnica más detallada.

- 0xC1900101 - 0x2000c
- 0xC1900101 - 0x20017
- 0xC1900101 - 0x30018
- 0xC1900101 - 0x3000D
- 0xC1900101 - 0x4000D
- 0xC1900101 - 0x40017

Asegúrese de que el dispositivo disponga de suficiente espacio. Se necesitan al menos 16 GB de espacio libre en el dispositivo para actualizar un sistema operativo de 32 bits o 20 GB para un sistema operativo de 64 bits. Para obtener más información, consulte liberar espacio de disco en Windows 10.

Ejecuta Windows Update varias veces. Descarga e instala las actualizaciones disponibles en Windows Update, incluidas las actualizaciones de software y hardware y algunos controladores de terceros. Usa el solucionador de problemas para Windows 10 para solucionar errores de Windows Update.

Comprueba los controladores de terceros y descarga las actualizaciones disponibles. Puedes encontrar controladores de terceros e instrucciones de instalación para cualquier hardware que hayas agregado al dispositivo en el sitio web del fabricante.

Desconecta el hardware adicional. Quita todas las unidades y dispositivos de almacenamiento externo, bases y cualquier otro hardware conectado al dispositivo que no sea necesario para la funcionalidad básica.

Compruebe si hay errores en el administrador de dispositivos. Selecciona el botón Inicio y, luego, en el cuadro de búsqueda de la barra de tareas, escribe Administrador de dispositivos. Elige Administrador de dispositivos en los resultados. En la ventana que se muestra, busca cualquier dispositivo con un signo de exclamación amarillo junto a él (puede que tengas que seleccionar cada categoría para cambiar a la lista de dispositivos). Pulsa y sostén (o haz clic con el botón derecho) en el nombre del dispositivo y selecciona Actualizar software de controlador o Desinstalar para corregir los errores.

Quita el software de seguridad de terceros. Asegúrate de que sabes cómo volver a instalar los programas y de tener a mano todas las claves de producto necesarias. Windows Defender te ayudará a proteger el dispositivo mientras tanto.

Repara los errores del disco duro. Selecciona el botón Inicio y, luego, en el cuadro de búsqueda de la barra de tareas, escribe símbolo del sistema. Elige Símbolo del sistema en la lista de resultados. En la ventana que aparece, escribe chkdsk/f C: y presiona la tecla ENTRAR. Las reparaciones se inician automáticamente en el disco duro, y se te pedirá que reinicies el dispositivo.

Restaurar y reparar archivos de sistema. Selecciona el botón Inicio y, luego, en el cuadro de búsqueda de la barra de tareas, escribe símbolo del sistema. Elige Símbolo del sistema en la lista de resultados. En la ventana que aparece, escribe DISM.exe /Online /Cleanup-image /Restorehealth y presiona la tecla ENTRAR. (Aprenda a reparar una imagen de Windows)

Otros errores comunes

Error	Qué significa y cómo solucionarlo
0xc1900223	Esto indica que se ha producido un problema al descargar e instalar la actualización seleccionada. Windows Update volverá a intentarlo más tarde y no necesitas hacer nada en este momento.
0xC1900208 – 0x4000C	Este error puede indicar que una aplicación incompatible instalada en el PC está impidiendo que se complete el proceso de actualización. Comprueba que todas las aplicaciones incompatibles estén desinstaladas y, después, intenta actualizar de nuevo.
0xC1900107	Una operación de limpieza de un intento de instalación anterior sigue pendiente y se requiere un reinicio del sistema para continuar con la actualización. Reinicia el dispositivo y vuelve a ejecutar el programa de instalación. Si el reinicio del dispositivo no resuelve el problema, usa la utilidad Liberador de espacio en disco y realiza una limpieza de los archivos temporales y de los archivos de sistema. Para obtener más información, consulta Liberador de espacio en disco en Windows 10.
0x80073712	Es probable que falte un archivo necesario para Windows Update o que esté dañado. Intente reparar los archivos del sistema: seleccione el botón **Inicio** y escriba **símbolo** del sistema en el cuadro de búsqueda de la barra de tareas. Elige **Símbolo del sistema** en la lista de resultados. En la ventana que aparece, escribe **DISM.exe /Online /Cleanup-image /Restorehealth** y presiona la tecla Entrar.
0xC1900200 – 0x20008 0xC1900202 – 0x20008	Este error puede significar que el PC no cumple los requisitos mínimos para descargar o instalar la actualización a Windows 10. Más información sobre los requisitos mínimos para Windows 10
0x800F0923	Esto puede indicar que un controlador u otro software de su equipo no es compatible con la actualización a Windows 10. Para obtener información sobre cómo solucionar este problema, póngase en contacto con el soporte técnico de Microsoft.
0x80200056	Este error puede indicar que el proceso de actualización se interrumpió porque se reinició el equipo o se cerró la sesión sin querer. Intenta la actualización de nuevo y asegúrate de que el equipo esté conectado y encendido.

0x800F0922	Este error puede indicar que el equipo no se puede conectar con los servidores de Windows Update. Si usas una conexión VPN para conectarte a una red de trabajo, desconéctate de la red, desactiva el software VPN (si procede) e intenta realizar de nuevo la actualización. Este error también puede indicar que no hay suficiente espacio libre en la partición reservada para el sistema. Este problema se puede solucionar usando software de otro fabricante para aumentar el tamaño de esta partición.
Error: No podemos completar las actualizaciones. Deshaciendo cambios, no apagues el equipo. Error: Error al configurar las actualizaciones de Windows. Revirtiendo cambios.	Estos son ejemplos de errores genéricos que pueden aparecer cuando una actualización de Windows produce un error. Tendrás que determinar el código de error específico para investigar cómo poder solucionar el problema de la mejor manera posible. Puedes encontrar el código de error de la actualización que ha fallado en el historial de actualizaciones. Busque la actualización que no está instalada, anote el código de error y, a continuación, póngase en contacto con el soporte técnico de Microsoft. Para ver el historial de actualizaciones en Windows 8.1: ▪ Para abrir Windows Update, desliza el dedo rápidamente desde el borde derecho de la pantalla (o bien, si usa un mouse, apunta a la esquina
Error: La actualización no se puede aplicar a tu equipo.	Este error puede indicar que tu PC no tiene las actualizaciones necesarias instaladas. Asegúrate de que todas las actualizaciones importantes estén instaladas en el PC antes de intentar actualizar.
0x80070070 – 0x50011 0x80070070 – 0x50012 0x80070070 – 0x60000	Este error puede indicar que el equipo no tiene suficiente espacio disponible para instalar la actualización. Libera espacio en la unidad y vuelve a intentarlo. Obtener sugerencias para liberar espacio en disco
0x80300024	El disco, partición o volumen de destino no admite la operación de disco especificada. Asegúrese de que su equipo cumple con los requisitos mínimos para instalar Windows 10.
0x80070002 0x20009	El sistema no encuentra el archivo especificado. Si tienes un disco o varios discos en los que no vas a instalar Windows 10, quítalos.
0xC1900101 0x20017	Un controlador ha provocado un problema.

0xC1900101 0x20017	Un controlador ha provocado un problema.
0xC1900101 0x30017	Deshabilita o quita todo el antivirus o antispyware de terceros del sistema. Desconecta todos los dispositivos periféricos que están conectados al sistema, excepto el mouse, el teclado y la pantalla. Ponte en contacto con el proveedor del hardware para obtener controladores de dispositivo actualizados.
0x8007042B 0x4000D	La instalación de Windows finalizó de manera inesperada debido a otro proceso que se ejecutaba en segundo plano. Al iniciar Windows en modo normal, varias aplicaciones y servicios se inician automáticamente y se ejecutan en segundo plano. Entre estos programas se encuentran procesos básicos del sistema, software antivirus, aplicaciones de utilidad del sistema y otro software que se haya instalado anteriormente. Estas aplicaciones y servicios pueden causar interferencias al intentar actualizar a la última versión de Windows 10. Para ayudarte a determinar si un programa en segundo plano está interfiriendo con la actualización, puede que sea necesario un "arranque limpio". Consulte Cómo realizar un inicio limpio en Windows.
0x800700B7 0x2000a	La instalación de Windows finalizó de manera inesperada debido a otro proceso que se ejecutaba en segundo plano. Desinstala cualquier software antivirus o antispyware y actualiza de nuevo.

BIBLIOGRAFIA

- Rendrán (2020). Sobre el uso de MLIR para Verona Recuperado el 27 de mayo de 2021 de: https://systemcall.eu/2020/10/22/on-using-mlir-for-verona/
- Microsoft (2020). Verona recuperado el 27 de mayo de 2021 de: https://github.com/microsoft/verona
- Microsoft (2019). Verona en Cambridge recuperado el 30 de marzo de 2021 de: https://www.microsoft.com/en-us/research/lab/microsoft-research-cambridge/
- Anónimo (2019). MLIR en Verona el 29 de mayo de 2021 recuperado de: https://systemcall.eu/2020/10/22/on-using-mlir-for-verona/
- Mathieu, Mihaela Juganaru. (2014) INTRODUCCION A LA PROGRAMACION. Primera edición, recuperado el 01 de mayo de 2021 https://editorialpatria.com.mx/pdffiles/9786074384154.pdf (PAGINA 13 DEL PDF)
- Valdeolmillos, Celia. (2020) Proyecto Verona: un nuevo lenguaje de programación creado por Microsoft e inspirado en Rusth. Recuperado el 03/05/2021 de ttps://www.muycomputerpro.com/2020/01/20/proyecto-verona-lenguaje-microsoft-rust
- Anonimo. (2020). Programador Clic Recuperado el 14 de mayo de 2021 de https://programmerclick.com/article/4158695469/
- Autor: Yiming, Jamin. (2020). Verona, un lenguaje de programación de código abierto de Microsoft Research, toma prestado características como Rust y Cyclone. Recuperado el 15 de mayo de 2021 de: https://programmerclick.com/article/4158695469
- VANESSA ROSSELLÓ VILLÁN. (2019). Las metodologías ágiles más utilizadas y sus ventajas dentro de la empresa. Recuperado el 16 de mayo de 2021 de: https://www.iebschool.com/blog/que-son-metodologias-agiles-agile-scrum/
- Calvo, Jorge. (2018). ¿Que es un Compilador en programación? Recuperado el 17 de mayo de 2021 de: https://www.europeanvalley.es/noticias/que-es-un-compilador-en-programacion/
- CILSA / Centro de Información y Atención Nacional. (2017). TECNOLOGÍA INCLUSIVA. ¿Qué es un programa? Recuperado el 18 de mayo de 2021 de: https://desarrollarinclusion.cilsa.org/tecnologia-inclusiva/que-es-un-programa/
- Anonimo. (2020). Wikipedia. Recuperado el 19 de mayo de 2021 de: https://en.wikipedia.org/wiki/Instruction_register

- Dialecto:
- Foro Wikipedia. Anónimo. Lenguaje de programación Recuperado el 20 de mayo de 2021 de: https://en.wikipedia.org/wiki/Programming_language#Definitions

- CODIGO ABIERTO
- Cristian Hernández. (2019) ¿Qué significa que un lenguaje de programación sea de código abierto? Recuperado el 21 de mayo de 2021 de: https://es.quora.com/Qu%C3%A9-significa-que-un-lenguaje-de-programaci%C3%B3n-sea-de-c%C3%B3digo-abierto

- ADMINISTRACION DE MEMORIA
- Anónimo. (2015). Rust (lenguaje de programación) Recuperado el 22 de mayo de 2021 de: https://www.udg.co.cu/cmap/sistemas_operativos/administracion_memoria/administracion_memoria/administracion_memoria.html

- RUST
- Anónimo. Foro Wikipedia. Recuperado el 23 de mayo de 2021 de: https://es.wikipedia.org/wiki/Rust_(lenguaje_de_programaci%C3%B3n)

- CYCLONE
- Anónimo. Foro Wikipedia. Cyclone (programming language) Recuperado el 24 de Mayo de 2021 de: https://en.wikipedia.org/wiki/Cyclone_(programming_language)

- SIMILITUDES
- Recuperado el 25 de mayo de 2021 de: https://www.chakray.com/es/lenguajes-programacion-tipos-caracteristicas/
- https://blog.buhoos.com/lenguajes-de-programacion-cuadro-comparativo/

- TIPOS DE MEMORIA:
- GuilleVen. (2017). Tipos de memorias de una computadora. Recuperado el 26 de mayo de 2021 de: https://www.tecnologia-informatica.com/tipos-memorias-computadora/

- ERRORES COMUNES:
- Microsoft. (2018). Obtener ayuda con los errores de actualización e instalación de Windows 10. Recuperado el 27 de mayo de 2021 de: https://support.microsoft.com/es-es/windows/obtener-ayuda-con-los-errores-de-actualizaci%C3%B3n-e-instalaci%C3%B3n-de-windows-10-ea144c24-513d-a60e-40df-31ff78b3158a

- Anónimo. (2020). On using MLIR for Verona Recuperado el 28 de mayo de 2021 de: https://systemcall.eu/2020/10/22/on-using-mlir-for-verona/

- INTRODUCCION:
- Anónimo. (2015). Nuestro compromiso es la educación. Recuperado el 29 de mayo de 2021 de: https://editorialpatria.com.mx/pdffiles/9786074384154.pdf (PAGINA 13 DEL PDF)

- OBJETIVO Y JUSTIFICACION
- Celia Valdeolmillos. (2020). Proyecto Verona: un nuevo lenguaje de programación creado por Microsoft e inspirado en Rust. Recuperado el 30 de mayo de 2021 de: https://www.muycomputerpro.com/2020/01/20/proyecto-verona-lenguaje-microsoft-rust

- objetivos especificos
- Yiming, Jamin. (2020). Verona, un lenguaje de programación de código abierto de Microsoft Research, toma prestado características como Rust y Cyclone. Recuperado el 31 de mayo de 2021 de: https://programmerclick.com/article/4158695469/

- Yiming, Jamin. (2020). Verona, un lenguaje de programación de código abierto de Microsoft Research, toma prestado características como Rust y Cyclone. Recuperado el 01 de Junio de 2021 de: https://programmerclick.com/article/4158695469/

- METODOLOGIA:
- Vanessa Rosselló Villán. (2019). Las metodologías ágiles más utilizadas y sus ventajas dentro de la empresa Recuperado el 02 de junio de 2021 de: https://www.iebschool.com/blog/que-son-metodologias-agiles-agile-scrum/

- COMPILADOR:
- JORGE Calvo. (2018). ¿Qué es un Compilador en programación? Recuperado el 03 de junio de 2021 de: https://www.europeanvalley.es/noticias/que-es-un-compilador-en-programacion/

- Programa:
- O.N.G por la inclusión. (2017) https://desarrollarinclusion.cilsa.org/tecnologia-inclusiva/que-es-un-programa/ Recuperado el 04 de Junio de 2021 de: https://desarrollarinclusion.cilsa.org/tecnologia-inclusiva/que-es-un-programa/

- IR:
- Anónimo. Foro Wikipedia. Instruction register Recuperado el 05 de Junio de 2021 de: https://en.wikipedia.org/wiki/Instruction_register

- Dialecto:
- Anónimo. Foro Wikipedia. Programming language. Recuperado el 06 de Junio de 2021 de: https://en.wikipedia.org/wiki/Programming_language#Definitions

- CODIGO ABIERTO
- Alejandro Hernández Ferriz, (2016). ¿Qué significa que un lenguaje de programación sea de código abierto?. Recuperado el 07 de Junio de 2021 de: https://es.quora.com/Qu%C3%A9-significa-que-un-lenguaje-de-programaci%C3%B3n-sea-de-c%C3%B3digo-abierto

- ADMINISTRACION DE MEMORIA
- Anónimo. (2018). Un nuevo lenguaje de programación Verona. Recuperado el 08 de Junio de 2021 de: https://www.udg.co.cu/cmap/sistemas_operativos/administracion_memoria/administracion_memoria/administracion_memoria.html

- RUST
- Anónimo. Foro Wikipedia. Rust (lenguaje de programación). Recuperado el 09 de Junio de 2021 de: https://es.wikipedia.org/wiki/Rust_(lenguaje_de_programaci%C3%B3n)

- CYCLONE
- Anónimo. Foro Wikipedia. Cyclone (programming language). Recuperado el 10 de Junio de 2021 de: https://en.wikipedia.org/wiki/Cyclone_(programming_language)

- SIMILITUDES
- Chakray, (2016). Lenguajes de programación: tipos y características Recuperado el 11 de Junio de 2021 de: https://www.chakray.com/es/lenguajes-programacion-tipos-caracteristicas/

- Alejandro Hernández Ferriz, (2016). ¿Qué significa que un lenguaje de programación sea de código abierto? Recuperado de 29 de abril de: https://blog.buhoos.com/lenguajes-de-programacion-cuadro-comparativo/

- TIPOS DE MEMORIA:
- GuilleVen. (2019). ¿Chequear la existencia de memoria cache en la PC?. Recuperado el 12 de Junio de 2021 de: https://www.tecnologia-informatica.com/tipos-memorias-computadora/

- ERRORES COMUNES:
- Microsoft. (2021). Correcciones de errores en general. Recuperado el 13 de Junio de 2021 de: https://support.microsoft.com/es-es/windows/obtener-ayuda-con-los-errores-de-actualizaci%C3%B3n-e-instalaci%C3%B3n-de-windows-10-ea144c24-513d-a60e-40df-31ff78b3158a

-