T.C. SANAYİ VE TEKNOLOJİ BAKANLIĞI

#MİLLİ TEKNOLOJİ HAMLESİ

TÜBİTAK

WEBASSEMBLY VE RUST

ARAŞTIRMA SERİSİ - SAYI 17

BİLGEM

YAZILIM TEKNOLOJİLERİ ARAŞTIRMA ENSTİTÜSÜ

Simge ve Kısaltmalar

Kısaltmalar	Açıklama
TÜBİTAK	Türkiye Bilimsel ve Teknolojik Araştırma Kurumu
BİLGEM	Bilişim ve Bilgi Güvenliği İleri Teknolojiler Araştırma Merkezi
YTE	Yazılım Teknolojileri Araştırma Enstitüsü
C	C Programlama Dili
C++	C++ Programlama Dili
CSV	Comma-Separated Values (Virgülle Ayrılan Değerler)
HTML	Hyper Text Markup Language (Köprü Metni Biçimlendirme Dili)
JS	JavaScript Programlama Dili
Rust	Rust Programlama Dili
WASM	WebAssembly

Yazar
Berat BAYRAM

Yayın Koordinatörü
Kübra ERTÜRK

Editörler
Sacit GÖNEN
Beyza ŞENEL
Tuğçe YILMAZ

Tasarım
Şeyma KOÇER

Bu sayımızdaki deneme çalışmaları için TÜBİTAK BİLGEM YTE '**WebAssembly**' teknoloji birliğine katkılarından dolayı teşekkür ederiz.

İletişim: 0(312) 289 92 22 - yte.bilgi@tubitak.gov.tr

https://bilgem.tubitak.gov.tr/yte/

İçindekiler

Ön Söz

TÜBİTAK BİLGEM Yazılım Teknolojileri Araştırma Enstitüsü (YTE), 2012 yılından bu yana yazılım teknolojilerinde Ar-Ge faaliyetleri yürüten bir araştırma kuruluşudur. Araştırma faaliyetlerinde elde ettiği birikimini stratejik, hassas ve kritik projeler yürüterek kamu adına hayata geçirmekte; kurumlarımıza dijital dönüşüm, yazılım geliştirme teknolojileri ve kalite süreçleri konusunda danışmanlık vermektedir.

TÜBİTAK BİLGEM YTE tarafından hazırlanan Araştırma Serisi ile kurum içi içerik üretme çalışmalarının yaygınlaştırılması ve hazırlanan içeriklerin sektörün erişimine açılması amaçlanmaktadır. Araştırma Serisi'nde yayınlanan çalışmalar TÜBİTAK BİLGEM YTE çalışanlarının projelerde elde ettiği bilgi birikimini paylaşmak adına derlenmiştir. Bu çalışmalar ile ülkemizin yazılım sektörüne katkı sağlanılması hedeflenmektedir.

Giriş

Dijital çağın ve modern dünyanın getirilerine bakıldığında, eskiden elle yapılan birçok iş artık çevrim içi ortamda gerçekleştirilebilmektedir. Örneğin, önceden kâğıt kalemle saatlerce sırada bekleyerek yapılan işlemler, web sitelerinde kolayca yapılabilmektedir. Benzer şekilde önceden mağazalardan alınan CD veya DVD'ler, medya siteleri aracılığıyla doğrudan cihazlara indirilebilmektedir. Hatta çok sık kullanılan birçok büyük masaüstü uygulaması bile artık web sitesi olarak hizmet vermektedir.

Gelişmiş ve karmaşık uygulamaların verimli bir şekilde hayata geçirilebilmesi için modern teknolojilerin yer aldığı bir teknoloji zinciri oluşturulması gerekmektedir. Günümüzde, bu teknoloji zincirinin son halkasında büyük ölçüde web tarayıcıları bulunmaktadır. Bu durumun bir sonucu olarak tarayıcılar artık sadece basit HTML görüntüleyiciler değil; aynı zamanda sanal gerçeklik içeriklerini oynatan, cihazların kameraları gibi donanımlarına erişebilen güçlü araçlar haline gelmişlerdir. Üstelik, tarayıcılar yeteneklerinin yanı sıra performanslarını da artırmışlardır. Önceden bilgisayarlarda bile uzun süre yüklenerek açılan masaüstü uygulamaları, artık cep telefonları tarayıcılarında anında açılır hale gelmiştir.

Bu performans gelişmelerinin önündeki en büyük engellerden biri, JavaScript dilinin tarayıcı tarafından yorumlanan bir dil olması ve bu nedenle web sitesini açan cihazın donanımının tam gücüne erişememesiydi. WebAssembly (WASM) ise tam olarak bu sorunu çözmek için icat edilen yeni ve modern web özelliklerinden biridir. Artık bu teknoloji sayesinde, masaüstü uygulamaları seviyesinde yüksek işlem gücü gerektiren web uygulamaları inşa edilebilmektedir. Ancak, bu yüksek kabiliyetli sistemi kullanmak, daha üst seviye çalışan JavaScript dilinin aksine, işlemciye çok daha yakın çalışan, işlemci seviyesinde kod yazmayı gerektirir. Özellikle bu sebepten ötürü WASM yazımında kullanılması için tasarlanan Rust programlama dili ortaya çıkmıştır.

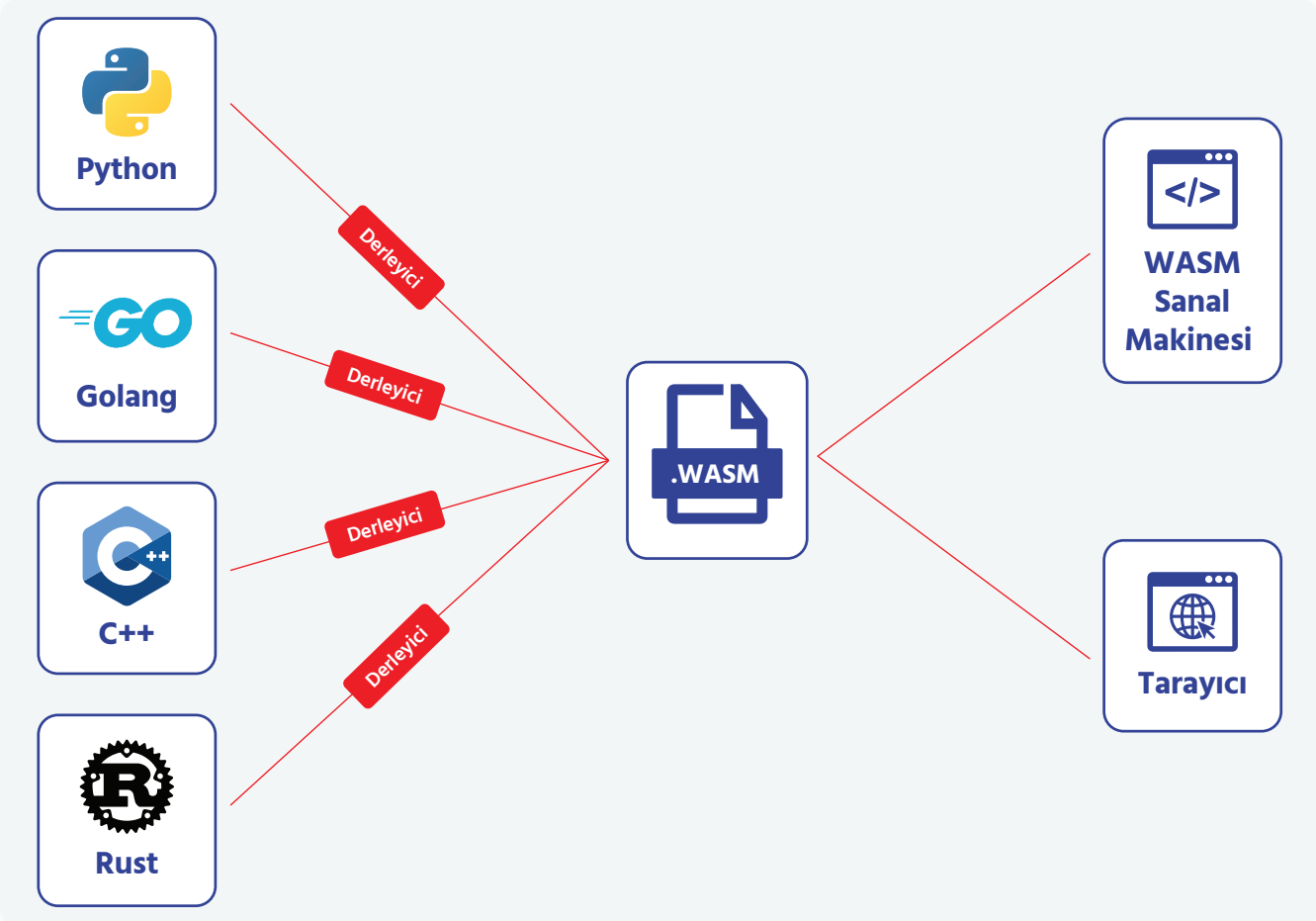
Bu çalışma kapsamında WebAssembly ve Rust dilinin özellikleri, kullanım alanları gibi konulara değinilmiş ve Proof of Concept (Kavram Kanıtı) çalışmaları aktarılmıştır.

WASM Nedir?

WebAssembly, yerel dillere yakın performans ile modern web tarayıcılarında ve ilgili çalışma ortamı (örneğin wasmer ya da wasmtime) olan tüm sistemlerde çalıştırılabilen kod türüdür. Web sayfala-rında C / C++ veya Rust gibi yüksek seviyeli dillerin derlenebilmesi, çalıştırılabilmesi ve taşınabilmesini sağlayan taşınabilir bir derleme hedefi olarak tasarlanmıştır. Ayrıca JavaScript ile uyumlu olarak tasarlanmış olup her ikisinin de birlikte çalışmasına olanak tanır.

Şekil 1'de görüldüğü gibi birçok program dili *.wasm formatına derlenebilir ve ortaya çıkan binary (ikili) içerik, çalıştırmak istediğimiz platformda uygun bir çalıştırma ortamı varsa çalıştırılabilir.

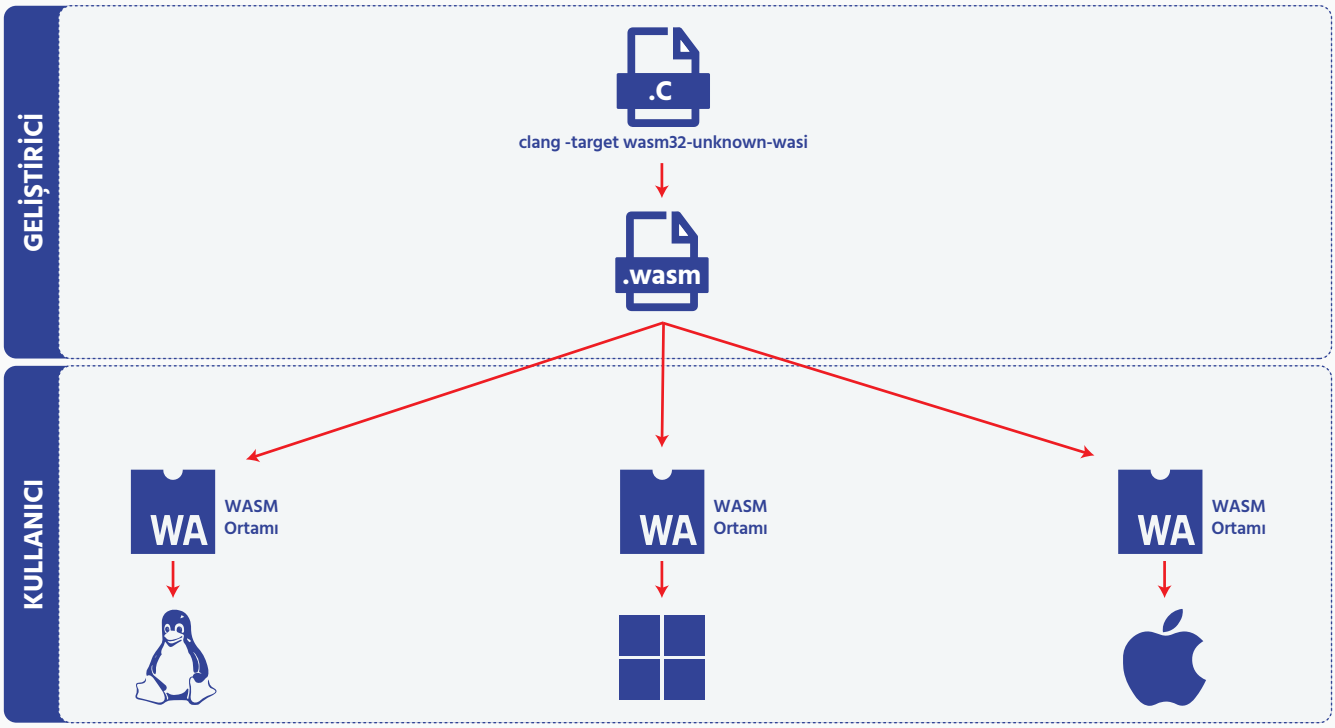
Java programlama dilinin de benzer bir amacı var ancak WebAssembly, Java'dan farklı olarak plat-form ile programlama dilini soyutlar. Bu sayede istenilen programlama diliyle geliştirme yapılabilir.



Şekil 1. Farklı Dillerin Sanal Makine ve Tarayıcılarda Çalışması

Akıllara "Zaten mevcut bir kodu farklı platformlarda çalıştırabiliyorduk." şeklinde bir düşünce gele-bilir. Örneğin; C dili ile yazılmış bir kodu farklı platformlar için derlenebilir. Ancak bu durumda her platform farklılığı için kod yazmak ve her biri için ayrı build (yapı) almak gereklidir.

Şekil 2'de görülebileceği gibi WebAssembly ise **tek bir build alarak** uygulamaların çalışabilmesini sağlamaktadır.

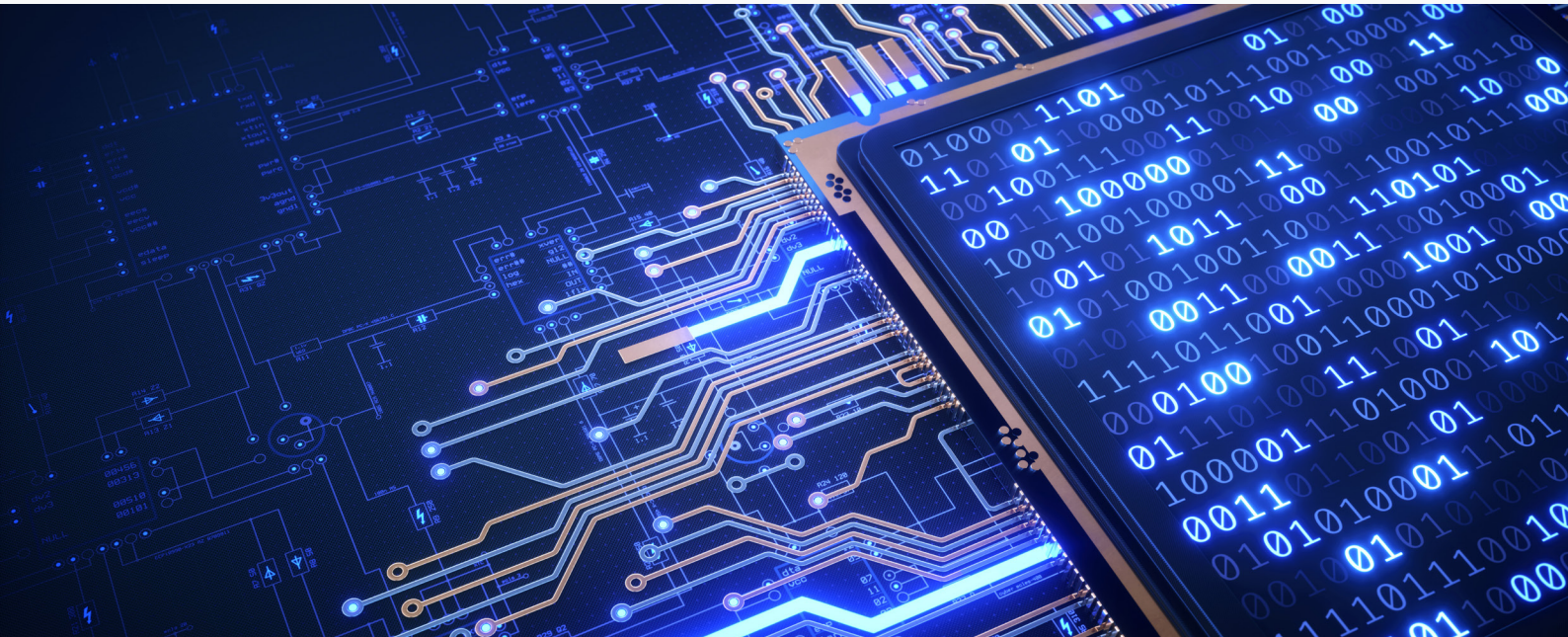


Şekil 2. Tek .Wasm Dosyasının Farklı İşletim Sistemlerinde Kullanılması

WASM'ın Tarihi

Birçok büyük firma, web tarayıcılarda yüksek performanslı uygulamalar yapmanın yollarını aradı. Bunların en büyük iki örneği Google'ın NaCl [Native Client, (Yerel İstemci)] teknolojisi ve Mozilla'nın asm.js kütüphanesidir. Ama hiçbiri bir masaüstü uygulaması kadar performanslı çalışmıyordu.

Bu problemi daha temelden çözebilmek için 2015 yılında Apple, Google, Microsoft ve Mozilla gibi birbirlerine rakip teknoloji firmaları, ortak bir web programlama standardı konusunda karar kılmak için bir araya gelerek "WebAssembly Community Group" adı altında birleştiler. Ana hedef; Flash Player, Unity Player, Silverlight gibi eklentiler kullanılmadan yüksek performans gerektiren işlemlerin gerçekleştirilebilmesiydi.



WASM'ın Avantajları

- **Verimli ve Hızlı:** WASM, boyut ve yükleme süresi açısından verimli bir binary formatta kodlanacak şekilde tasarlanmıştır. Birçok platformda mevcut olan ortak donanım özelliklerinden yararlanarak yüksek hızda çalıştırmayı amaçlamaktadır.
- **Güvenli:** Her WASM modülü dışarı ile tamamen izole olan bir sandbox (kum havuzu) ortamında çalışır. Bu sayede;
  - Uygulamalar bağımsız ama belirlenen sınırlar içinde çalışır ve uygun API'ler kullanılmadığı taktirde sandbox'tan çıkamaz.
  - Her modül gömülü olduğu ortamın güvenlik politikalarına tabidir. Örnek politikalardan bazıları web tarayıcılar için CORS, Unix sistemler için POSIX'dir.
  - Bir WASM modülünün neye erişebileceğini modülü çalıştıran ortam kontrol eder (mobil platformlardaki gibi kapsamlı ve detaylı kontrol).
- **Açık Web Platformunun Bir Parçası:** WASM, web'in sürümsüz, özellikleri test edilmiş ve geriye dönük uyumlu yapısını korumak için tasarlanmıştır. Herhangi bir web sayfası, ne kadar zaman geçerse geçsin, nasıl bir anda çalışmayı bırakmıyorsa (bkz. www.spacejam.com/1996/) bir WASM modülü de hiçbir zaman çalışamaz hale gelmeyecektir.

WASM'ın Kullanım Örnekleri

Web, yalnızca belgelere uygun bir platform olarak başladı ancak tarihi boyunca önemli ölçüde büyüdü. Gmail gibi ilk uygulamalar, daha karmaşık etkileşimin ve uygulamaların mümkün olabileceğini gösterdi. İlk uygulama zamanlarından bugüne kadar, web uygulamaları sınırları zorladıkça, tarayıcılar da yeteneklerini genişleterek bu yeni talebe yanıt verdi.

Aşağıda bahsedilen örneklerin dışındaki birçok değerli projeye madewithwebassembly.com sitesinden ulaşılabilir.

Adobe Photoshop Çevrimiçi

Photoshop kadar karmaşık bir yazılımı doğrudan tarayıcıda çalıştırma fikrini birkaç yıl önce hayal etmek oldukça zordu ancak bugün mümkün. Adobe önceki denemelerinde Spark ve Lightroom ürünlerini web'e getirmişti ve Photoshop'u da aynı şekilde tarayıcılara taşımak istiyordu. Ancak JavaScript'in performans limitlerine takılıyorlardı. Emscripten ve WASM ise bu limitleri kaldıran anahtarlardı.

Emscripten, C++ kodunuzu WASM'a çeviren bir araçlar zinciridir. Bu zincir, sadece çeviri ile kalmayıp, ek olarak POSIX API çağırılarını Web API çağırılarına ve hatta OpenGL'i WebGL'e bile dönüştürebilir. Bu kolay çeviri gücü özellikle Adobe'un elini çok rahatlattı çünkü Emscripten sayesinde yıllar boyunca geliştirilmiş Photoshop kodlarını baştan yazmalarına artık gerek yoktu. İşin performans tarafının da WASM ile çözülmesi sayesinde Adobe Photoshop Çevrimiçi kullanıcılara açılmış oldu.

Google Earth

İlk Google Earth yine C++ ile yazılmış bir masaüstü uygulamasıydı. Mobil çağın yükselmesiyle beraber; Google Earth, Objective-C++ ve Android NDK [Native Development Kit, (Yerel Geliştirme Kiti)] ile iOS ve Android'e getirildi. 2017 yılında ise, Google'ın kendi geliştirdiği ve yalnızca Chrome içinde çalışan Native Client (NaCl) ile sadece kendi tarayıcısına getirmeyi başardı ama performansı yeterli değildi. Google Earth, WebAssembly ile tüm tarayıcılarda aktif hale gelirken performansı da arttı.

Figma

WebAssembly gelmeden önce Figma kendi C++ kodlarını asm.js ile çalıştırıyordu. asm.js ise C / C++ dilini JS ortamı içinde taklit eden bir JS alt dilidir. Yani mevcut C / C++ kodları asm.js ile JS'e dönüştürülerek web tarayıcılar içinde kullanılabiliyor. Ancak JS doğası gereği alt seviye dillerden daha yavaş olduğu için asm.js kullanılan uygulamalarda performans sıkıntıları oluşuyordu.

WebAssembly ise kullanım olarak asm.js'in bire bir aynısı ama oluşan kod makine seviyesine daha yakın çalıştırıldığı için çok daha hızlı. Sonuç olarak, Figma'nın WASM kazanımları şu şekilde sıralanabilir:

- Oluşan format çok sıkı olduğu için daha az internet trafiği vardır.
- **Yaklaşık 20 kat** daha hızlı çözümleme sunar.
- C++ kodu LLVM tarafından zaten optimize edildiği için tarayıcı direkt kodu çalıştırabilir (JS tarafında, kod direk çalışmadan önce optimize edilmesi gerekir.).
- Tarayıcılar WASM'ın native (yerel) koda dönüşümünü önbellekleyebilir. Yani aynı uygulama ilk seferden sonra çok daha hızlı açılabilir (Aynı durum asm.js'de söz konusu değildir.).
- WebAssembly 64-bit tam sayılara direkt destek verir [JS sadece 53-bit tam (64-bit ondalık) destekleyebilmektedir.].

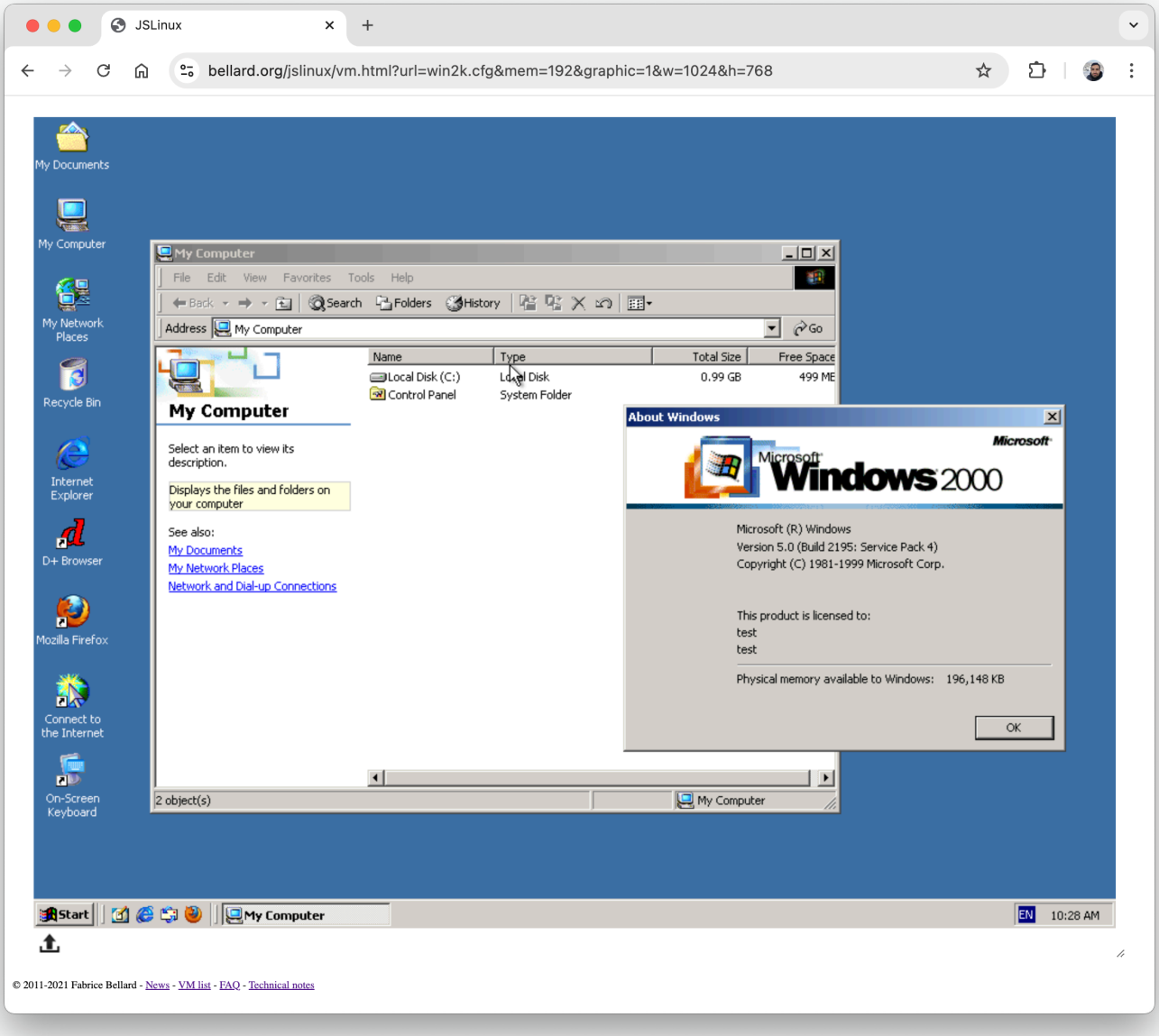
Microsoft

Microsoft'un yönettiği .NET ekosistemi C / C++ / C# üzerinden dönmektedir ve bu ekosistem sadece masaüstü uygulamalara odaklıydı. Bu nedenle ASP.NET takımı, güçlü ekosistemlerini tarayıcılara

getirebilmek amacıyla içinde WASM kullanan Blazor teknolojisini geliştirdiler. Blazor ile geliştiriciler uygulama geliştirirken, JavaScript yazmak yerine C# ile interaktif web uygulamaları geliştirebilir hale geldiler. Ayrıca Blazor, F# ve Visual Basic'i desteklemekte ve bu geniş destek sayesinde daha önce .NET teknolojilerinde uğraşmış bir geliştirici, komple yeni bir ekosistem öğrenmeden web'e yönelik geliştirmeler yapabilir.

Microsoft'un gelecek hedeflerinden biri ön ve arka yüzde farklı teknolojiler kullanmak yerine bir web uygulamasını baştan sonra .NET ile kurgulayabilmek ve WebAssembly bu yoldaki en önemli teknolojilerden biridir.

JS Linux



Şekil 3. Tarayıcı İçinde Olan JS Linux İçinde Windows 2000 İşletim Sistemi

Artık, bir işletim sisteminin içindeki tarayıcıda bir işletim sistemi çalıştırarak, bu işletim sistemi içinde bir tarayıcıyı çalıştırmak mümkün. Açık kaynak dünyasındaki birçok önemli aracın (ffmpeg, tiny-c, QEMU, vs.) geliştiricisi olan Fabrice Bellard tarafından yazılan **TinyEMU**; RISC-V ve x86 mimarileri için bir emülatör. Şekil 3'de ekran görüntüsü bulunan JS Linux ise bu emülatörün; Emscripten kullanılarak WASM ve JS koduna dönüştürülüp, tarayıcıda çalıştırıldığı versiyonudur. Bu web sitesi sayesinde Windows 2000 ve Fedora 33 gibi birçok işletim sistemi hiçbir kurulum olmadan basitçe tarayıcı içinde deneyimlenebilir.

AutoCAD Web App ve Google Ink

AutoCAD web uygulaması, 35 yılı aşkın C / C++ kodunu web'e taşımak için Emscripten kullanmaktadır. WebAssembly'nin gücü sayesinde bu tarz yüksek performans isteyen uygulamalar tarayıcıda çalışabilmektedir.

Benzer şekilde Google'ın "Chrome Canvas", "Keep" ve "Chrome PDF" içinde kullandığı serbest çizim kütüphanesi olan "Google Ink", web platformunda da kullanabilir.

Tensorflow.js

Yapay zekâ geliştirme sistemleri donanımı tam gücü ile kullanmak isteyen uygulamalardır ve bu uygulamaların yerel versiyonlarındaki performanslarına JS ile erişmek pek mümkün değildir. Bu nedenle meşhur yapay zekâ kütüphanesini tarayıcılara getirmek isteyen TensorFlow ekibi, WebAssembly kullandıklarında yaklaşık 10 kat hız artışı gözlemlediler.

Oyun Sektörünün Geleceği

Günümüzde büyük çapta oyunları hemen oynayabilmek kolay bir iş değildir. Çoğunlukla bir başlatıcının kurulması ve 200 GB'lara varan indirmelerin yapılması gerekmektedir. Bazı oyunlar sadece konsollara özel, bazı oyunlar ise sadece belirli işlemci ve/veya ekran kartları ile çalışabilmektedir. Flash Player'ın sektörde baskın olduğu dönemdeki gibi basitçe bir siteye gidilip en büyük oyunların bile hemen oynanabilmesi sektörde ulaşılmaya çalışılan hedeflerden biridir.

Bu hedefe ulaşabilmek için sektörde birçok denemeler baş göstermeye başladı. Continuation Labs; Bethesda'nın meşhur oyunu DOOM 3'ü, WASM ile tarayıcılara getirmeye başardı. Popüler oyun motoru Unity, 5.6 sürümünden itibaren WebAssembly olarak çıktı verebiliyor. Benzer şekilde Wonder Interactive, Unreal Engine çıktılarının web'de çalıştırabilir hale getirilebilmesi için çalışmalarını sürdürüyor.

Rust

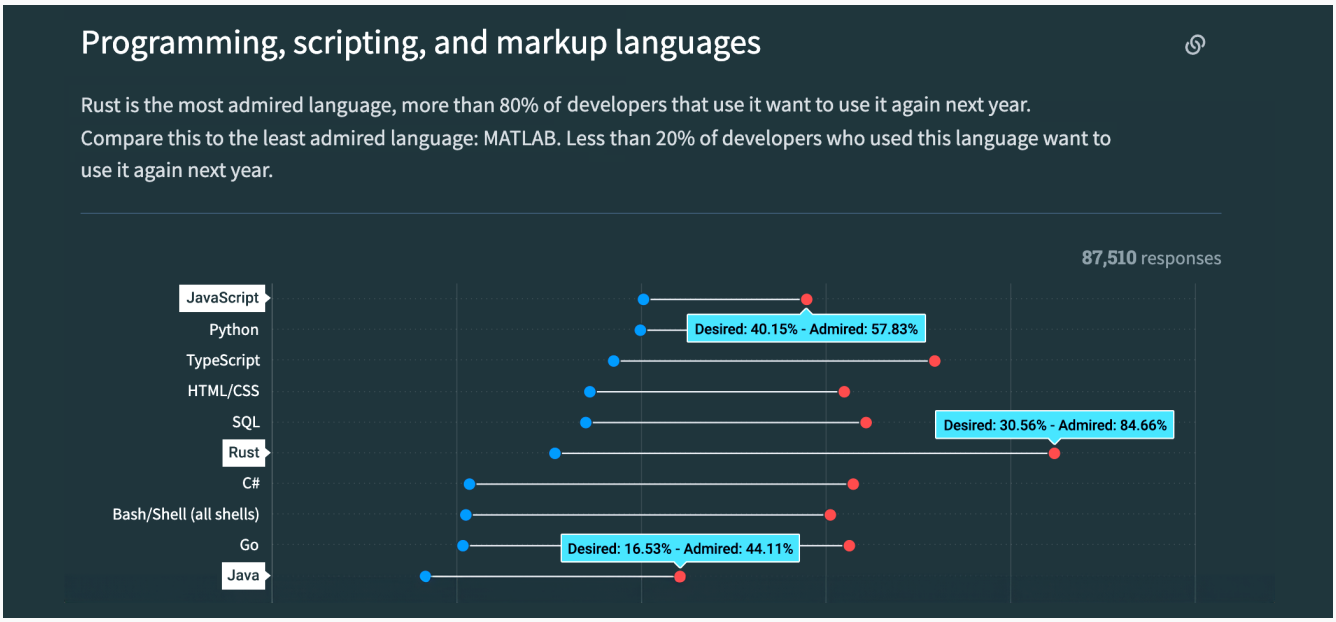
Bahsedilen birçok örnek önceden var olan C / C++ kodlarını tarayıcıya getirmeyi içeriyordu. Peki bugün sıfırdan bir WebAssembly projesine başlanacak olsa hangi teknolojiler kullanılmalı? Günümüzde bu iş için en çok tercih edilen dil ise Mozilla'nın kuruculuğunu yaptığı Rust dilidir.

```
fn factorial (i: u64) → u64 {
  if i == 0 {
    1
  } else {
    i * factorial(i - 1)
  }
}
```

Şekil 4. Rust Dilinde Faktöriyel Hesaplayan Fonksiyon Örneği

Mozilla, Firefox tarayıcısındaki performans ve bellek açığı problemlerini giderebilmek için yeni bir tarayıcı motoru olan Servo üzerinde çalışıyordu. Aynı durum ileride tekrar etmesin diye, C++ dilinin kırılgan yapısından kurtulmak isteniyordu. Bu amaçla hem C++ gibi hızlı çalışan hem de kötü kod yazmayı oldukça zor kılan Rust dili geliştirilmeye başlandı. 2021 yılında ise AWS, Huawei, Google, Microsoft ve Mozilla'nın da içinde olduğu Rust Foundation kuruldu ve bu kurul tarafından gelişimi devam ediyor. Rust ile yazılmış basit bir faktöriyel fonksiyonu Şekil 4'den incelenebilir.

Rust; performansı, tip güvenliliğini ve eş zamanlılığı vurgulayan çok paradigmalı, genel amaçlı bir programlama dilidir. Temel hedefi ise sistem korunurluğunu ve sürdürülebilirliğini sağlarken hız ve verimden ödün vermemektir. Şekil 5'de görülebileceği gibi; popüler soru-cevap sitesi olan Stack Overflow'un 2023 yılında yaptığı geliştirici anketinde, sektörde yaygınlığı az da olsa en çok takdir edilen dil Rust seçildi.



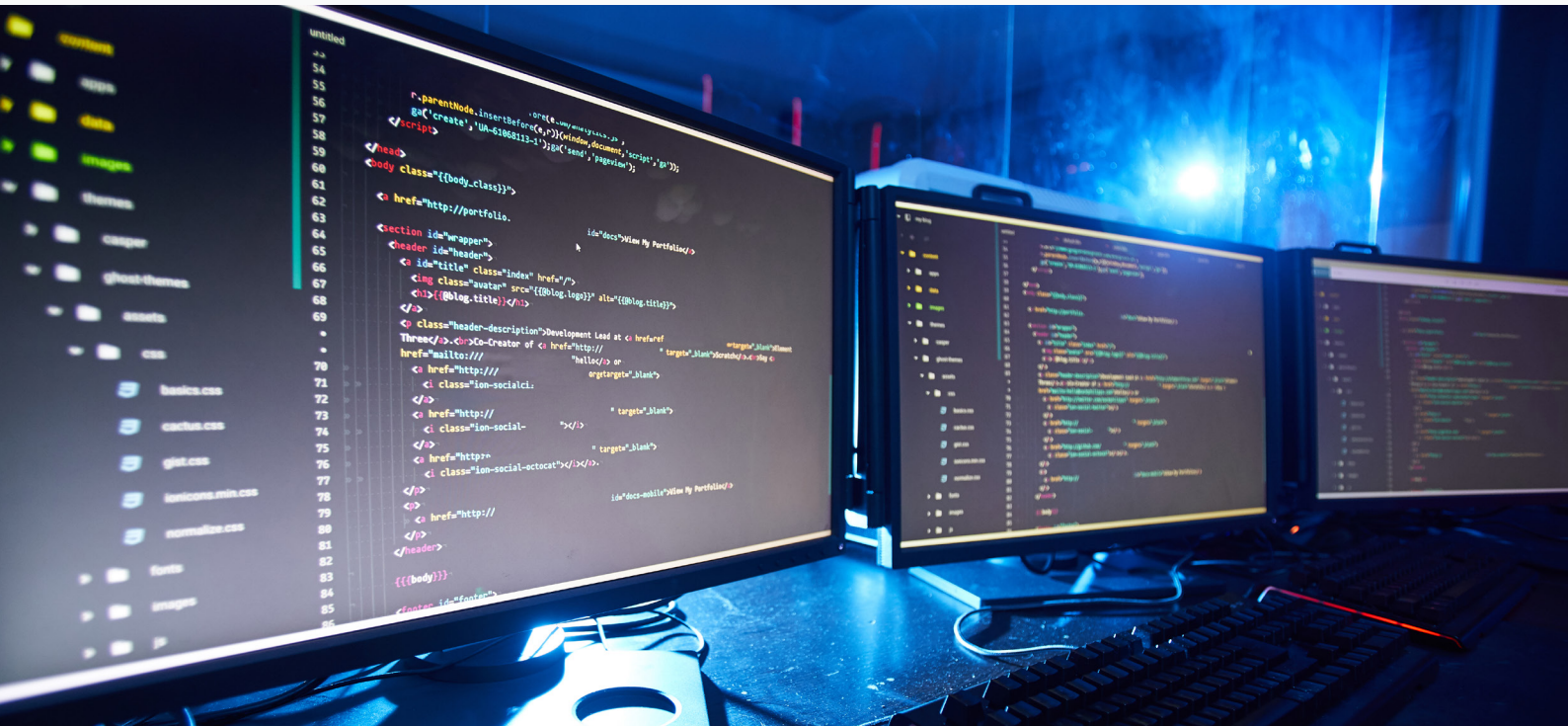
Şekil 5. Stack Overflow Geliştirici Anketi'ndeki JS, Rust ve Java Dillerinin Durumları

Özellikleri Nelerdir?

```
   Compiling error-messages v0.1.0 (/Users/ep/src/rust/error-messages)
error[E0277]: expected a `std::ops::FnMut<(char,)>` closure, found `std::string::String`
 --> src/lib.rs:3:29
  |
3 |     println!("{:?}", s.find("".to_owned()));
  |                        ^^^^^^^^^^^^^
  |                        |
  |                        expected an implementor of trait `std::str::pattern::Pattern<'_>`
  |                        help: consider borrowing here: `&"".to_owned()`
  |
  = note: the trait bound `std::string::String: std::str::pattern::Pattern<'_>` is not satisfied
  = note: required because of the requirements on the impl of `std::str::pattern::Pattern<'_>` for `std::string::String`

error: aborting due to previous error

For more information about this error, try `rustc --explain E0277`.
error: could not compile `error-messages`.

To learn more, run the command again with --verbose.
```

Şekil 6. Rust Hata Mesajı Örneği

- Geliştiricinin elinden tutan hata mesajları vardır (Bkz. şekil 6).
- Açık kaynak kodludur.
- Syntax (söz dizimi) olarak C++'a yakındır.
- C++ gibi dillerle kıyaslanabilir performans sağlar.
- Yeni sayılabilecek Go dilinde olduğu gibi Rust dili; Python ve Java gibi daha eski dillerin aksine OOP paradigmalarını doğrudan destekleyen yapılar bulundurmaz.
- WebAssembly yazmak için özellikle uygundur.
- Mızmızlığı ile ünlüdür. Güvensiz olan bir kod yazmak çok zordur. Bunun için özel makrolar (unsafe {} yapısı gibi) ile çok zorlanması gerekir.
- Otomatik bellek yönetimi ile bellek sızıntıları ve asılı referanslar gibi hataları önler.
- Çoklu iş parçacığı (thread) desteği sayesinde eş zamanlı programlama için güvenli ve verimli araçlar sunar.



Sahiplik Kontrolü

Sahiplik kontrol (borrow checker) modeli bellek güvenliğini zorunlu kılar, yani otomatik bellek yönetimi tekniklerinin kullanılmasını gerektirmeden tüm referansların geçerli belleğe işaret edip etmediğini derleme sırasında kontrol eder. Bu sayede garbage collector (çöp toplayıcı) veya runtime'a (çalışma ortamı) ihtiyaç duymaz.

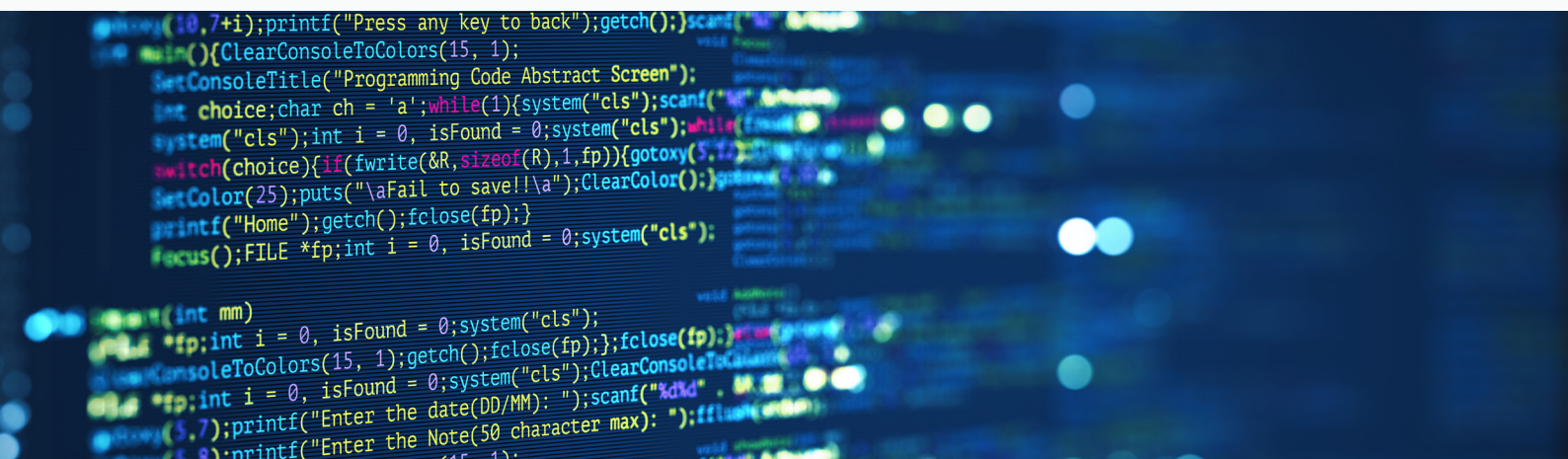
Rust'ın sahiplik sistemi, hafıza güvenliğini sağlayan katı kurallardan oluşur. Çalışırken her değerin tam olarak bir sahibi olması gerekir ve değerler, atama yoluyla veya bir değerin işlev parametresi olarak iletilmesi yoluyla farklı sahipler arasında taşınırlar. Değerler ödünç de alınabilir, yani sahibine iade edilmeden önce geçici olarak farklı bir işleve aktarılabilirler.

Bu kurallarla Rust, asılı referans (dangling pointer) sorununu çözer. Bir yığın veya geçici değişken kapsam dışına çıktığında, destructor'ı (imha edici) çalıştırılarak drop edilir (düşürülür). drop() fonksiyonu kodda manuel olarak da çağırılabilir ama derleyici bunu otomatik yapar.

```
fn yazdir_dize(s: String) {
    println!("{}", s);
}

fn main() {
    let s = String::from("Merhaba, Dünya!");
    yazdir_dize(s); // s, yazdir_dize tarafindan "tüketildi"
                    // bu nedenle artik kullanılamaz
    // baska bir yazdir_dize(s); derleme hatasiyla sonuçlanır
}
```

Şekil 7. Rust Sahiplik Yapısını Anlatan Kod Örneği

Sahiplik yapısı yukarıdaki Şekil 7'de verilen örnek ile daha rahat anlaşılabilir. Resimdeki Rust kodu, "Merhaba, Dünya!" mesajını yazdıran basit bir programdır. Örnekte s değişkenine "Merhaba, Dünya!" yazısı atanmış ve yazdir_dize fonksiyonuna gönderilmiştir. Bu değişken gönderildiği anda sahibi artık o fonksiyondur ve o fonksiyondan sonra o değişkene tekrar erişilemez. Örneğin, son yorum satırında olduğu gibi s değişkeni tekrar kullanılmaya kalkışılırsa sistem hata verecektir. Çünkü yazdir_dize fonksiyonu o değişkeni içinde silmiş olabilir, yani o fonksiyonun dışındaki kullanımının güvenliği garanti edilemez.



```
// Bu işlev bir kutunun sahipliğini alır ve onu yok eder
fn yok_et_kutu_i32(kutu_i32: Box<i32>) {
  println!("{} içeren kutuyu yok ediyor", kutu_i32);
}

// Bu işlev bir i32 değerini ödünç alır
fn odunc_al_i32(odunc_alinan_i32: &i32) {
  println!("Bu tam sayı: {}", odunc_alinan_i32);
}

fn main() {
  // Yığında bir kutulu i32 ve yığın üzerinde bir i32 oluştur
  // Hatırlatma: okunabilirlik için sayılara rastgele alt çizgiler eklenebilir
  // 5_1i32, 5i32 ile aynıdır
  let kutu_i32 = Box::new(5_i32);
  let yigin_i32 = 6_i32;

  // Kutunun içeriğini ödünç al. Sahipliği alınmaz,
  // bu nedenle içeriği tekrar ödünç alınabilir.
  odunc_al_i32(&kutu_i32);
  odunc_al_i32(&yigin_i32);

  {
      // Kutunun içindeki değere bir referans al
      let _i32_referansi: &i32 = &kutu_i32;

      // Hata!
      // Daha sonra kapsamda iç değer yok edilirken `kutu_i32` yok edilemez.
      yok_et_kutu_i32(kutu_i32);
      // FIXME ^ Bu satırı sil

      // İç değer yok edildikten sonra '_i32_referansi'nı ödünç alma girişimi
      odunc_al_i32(_i32_referansi);
  } // _i32_referansı kapsam dışına çıkar ve artık ödünç alınmaz.

  // 'kutu_i32' artık 'yok_et_kutu_i32'ye sahipliğini devredebilir ve yok
edilebilir
  yok_et_kutu_i32(kutu_i32);
}
```

Şekil 8. Rust Sahiplik Yapısını Anlatan Daha Kapsamlı Kod Örneği

Şekil 8'deki kod, bir i32 değerini, yığın üzerinde nasıl kutulayacağını ve ödünç almayı gösteren bir örnektir. Kod şu adımları izler:

1. **Kutulama:** yok_et_kutu_i32 işlevi, i32 değerinin sahipliğini alan ve onu yok eden bir fonksiyondur. odunc_al_i32 işlevi ise i32 değerini sadece ödünç alan bir fonksiyondur.
2. **Değer Yaratma:** main fonksiyonu başladığında, kutu_32 adlı bir kutu oluşturulur ve 5_i32 değerine atanır. Ayrıca, yigin_32 adlı bir değişken oluşturulur ve 6_i32 değerine atanır. Bu, yığın üzerinde değil, doğrudan Stack (yığın) üzerinde ayrılan bir tam sayı değeridir.
3. **Ödünç Alma:** Daha sonra, odunc_al_i32 işlevi her iki i32 değerine de çağrılır. İlk olarak, kutu_32 değerinin sahipliği alınmadan kutusundan ödünç alınır. Bu, kutunun içindeki değerin bir kopyasını oluşturmak yerine, var olan değere bir referans oluşturur. Ardından, yigin_32 değerine de aynı şekilde ödünç alınır.
4. **Blok Kapsamı Sorunu:** Kodun gövdesinde bir blok ({...}) bulunur. Bu bloğun içinde, kutu_32 değerine bir referans olan _i32_referansı adlı bir değişken oluşturulur. Ancak, hemen sonra yok_et_kutu_i32 işlevi çağrılmaya çalışılırsa, Rust bir hata verecektir. Bu, bloğun kapsamı sona erdiğinde _i32_referansı değişkeni silinse de kutu_32 değerine hala bir referans olması nedeniyle olur. Rust, sahipliği güvenli bir dil olduğundan, bir değerin sahipliği devredilmeden veya kopyalanmadan yok edilmesini önler.
5. **Kutunun Yok Edilmesi:** Bloktan sonra, yok_et_kutu_i32 işlevi artık kutu_32 değerini güvenli bir şekilde çağırabilir çünkü ödünç alınan herhangi bir referans bloğun kapsamı sona erdiğinde silinmiştir. Bu işlev, kutu_32 değerinin sahipliğini devralır ve kutuyu yok eder.

Rust Geliştirme Araçları

Rust ekosistemi, geliştirme deneyimini geliştirmeye yardımcı olabilecek çok çeşitli araçlara sahiptir. Bunlardan birkaçı şunlardır:

Kurulum Araçları:

- **Rustup:** Rust'ı yüklemek ve yönetmek için kullanılan paket yöneticisi ve sürüm aracıdır.
- **Cargo:** Rust projelerini oluşturmak, derlemek, test etmek ve çalıştırmak için kullanılan paket yöneticisi ve yapı aracıdır.

Entegre Geliştirme Ortamları (IDE'ler):

- **Visual Studio Code:** Rust için en popüler IDE'lerden biridir. Ücretsiz, açık kaynak kodludur ve çok çeşitli eklentiler sunar.
- **RustRover:** Ücretli ve kapalı kaynak kodludur. Ünlü java IDE'si IntelliJ IDEA ve JS IDE'si Webstorm'un geliştiricisi olan JetBrains firması tarafından geliştirilmiştir.

Ek Araçlar

- **rustfmt:** Rust kodunu otomatik olarak biçimlendirmek için kullanılır.
- **cargo test:** Rust kodunu test etmek için kullanılır.
- **miri:** Rust kodunu statik olarak analiz etmek ve bellek güvenliği hatalarını bulmak için kullanılır.
- **gdb:** Rust kodunda hata ayıklamak için kullanılır.

Rust'ın Sektörde Kullanım Örnekleri

Rust ile yazılan birçok yazılım vardır:

- Firefox Quantum içindeki Servo motoru,
- Cisco'nun OpenDNS servisi,
- Cloudflare güvenlik duvarlarındaki desen eşleme işlemleri,
- Açık kaynak blok zinciri platformu Polkadot,
- JavaScript ekosistemindeki çıkan birçok yeni araç (Deno, Turbopack, Rspack).

Microsoft, Windows'un bazı parçalarını Rust ile tekrar yazmaya başladığını duyurmuştur. Ayrıca diğer birçok kullanıma github.com/omarabid/rust-companies sitesinden bakılabilir.

Neden WASM için Rust Kullanmalıyız?

WASM yapısı gereği teknik olarak birçok programlama dili ile yazılabilir. Üretilen .wasm dosyası kendi başına ayağa kalkabilir olması gerektiği için Java/C#/Python/JS gibi yorumlayıcısı olan diller ile yazılması ideal değildir. Çünkü o yorumlayıcının .wasm dosyasına eklenmesi gerekmektedir.

Aslında C++ ve Go, WASM için oldukça mantıklı diller olsa da kendi araçları sıfırdan yazılacak olan WebAssembly uygulamaları için yeterli değildir. Mevcut büyük projeleri taşımak için yine de kullanılabilir.

Rust'ın geliştirilmesinin amaçlarından biri olması ve kendi araçlarının bunu hedef alarak geliştirilmiş olması, onu WASM için özellikle uygun kılmaktadır.

WASM/Rust Geliştirmenin Dezavantajları Nelerdir?

Hem WASM hem de Rust ekosisteminin en büyük dezavantajı diğer birçok köklü ekosisteme göre görece yeni olmasıdır. Her yeni yazılımda oluşan aşağıdaki durumlar WASM/Rust'ı da etkilemektedir:

- **Az kaynak:** Rust'ın kendi dokümantasyonu yeterli olsa da dış eğitimler ve Türkçe kaynak genel olarak oldukça azdır. Bu durum geliştirmeye yeni başlayacak ve İngilizce bilmeyen kişiler için büyük bir sıkıntı olabilir.
- **Az hazır kütüphane:** Mevcut hazır kütüphane miktarı azdır. Bu sebepten dolayı çoğu temel özelliğin elle yazılması gerekebilir.
- **Olgunlaşmamış çerçeveler (framework):** Mevcut çerçeve yapılarındaki özellik sayısı sınırlıdır.
- **Test ve hata ayıklama yapıları oturmuş değil:** Bu durum yeni yazılımların geliştirme süresini kötü etkileyebilir.
- **Platform API'lerine erişim stabil değil:** WASI gibi WASM ile bulunduğu platformlar arasındaki iletişimi sağlayan çerçeveler tüm mevcut işletim sistemlerinde standartlaştırılmış değil. Bu sebepten dolayı farklı cihazlarda aynı deneyimi sağlamak zor olabilir.
- **Küçük Topluluk:** C++ ya da JS gibi geniş bir geliştirici sayısına ve geliştirme tarihine sahip değil. Bu sebepten dolayı birçok problemle ilk karşılaşılan olunma ihtimali ve kişinin tek başına yardım almadan çözmek zorunda kalma ihtimali yüksektir.

Bu makalenin yayınlandığı tarih itibarı ile bir projeyi tamamen bu teknolojilere adamak yerine çok performans gerektiren küçük işlev parçalarıyla başlanması daha mantıklı olabilir.

Deneme (Proof of Concept) Çalışmaları

WASM ile Tarayıcıda Alert()

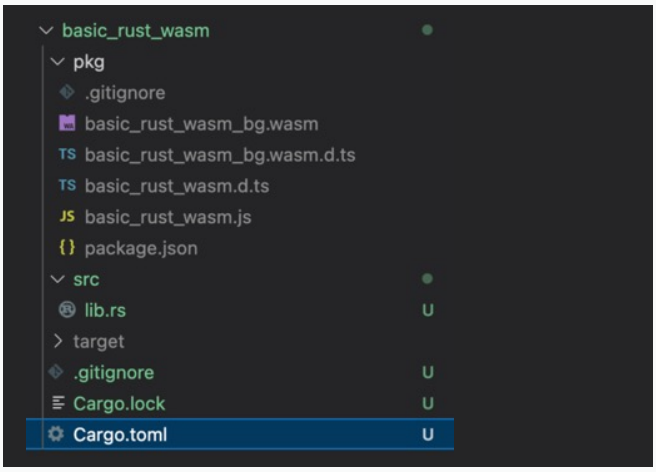
Deneme çalışmaları kapsamında javascript kullanmadan rust ve C++ üzerinden tarayıcıda alert() gösterilmeye çalışıldı. Bu çalışmalarla sistem fonksiyonlarına erişimin test edilmesi amaçlandı.

Rust

Platform API'lerine erişim Rust üzerinden test edildi. Rust'ın şu an tarayıcı API'lerine doğrudan erişim imkânı olmadığından, C++ binding'i (bağlantı) aracılığıyla tarayıcının kendi alert() fonksiyonu kullanıldı. Basit bir Rust projesi içerisinde derlenen WASM kodu ve WASM çıktısı elde edildi. Bu çıktının web tarayıcısında kullanılması ve butona basılması ile tarayıcının standart uyarı panelinin çalıştığı gözlemlendi. Şekil 9, 10 ve 11'de deneme sürecinde kullanılan kodlar ve kodların çıktıları gösterilmiştir.

```rust
use wasm_bindgen::prelude::*;

#[wasm_bindgen]
extern "C" {
    fn alert(s: &str);
}

#[wasm_bindgen]
pub fn show_alert() {
    alert("Hello World!");
}
```

Şekil 9. Rust ile WASM "Hello World!" Kodu



Şekil 10. Rust Kodunun Derleme Çıktıları

```html
<!doctype html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <title>Rust Alert</title>
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  </head>
  <body>
    <button id="my-button">Alert At</button>
    <script type="module">
      import init from './basic_rust_wasm.js';

      const wasmModule = await init(' ./basic_rust_wasm_bg.wasm');
      const myButton = document.getElementById('my-button');
      myButton.addEventListener('click', () => {
        wasmModule.show_alert();
      });
    </script>
  </body>
</html>
```

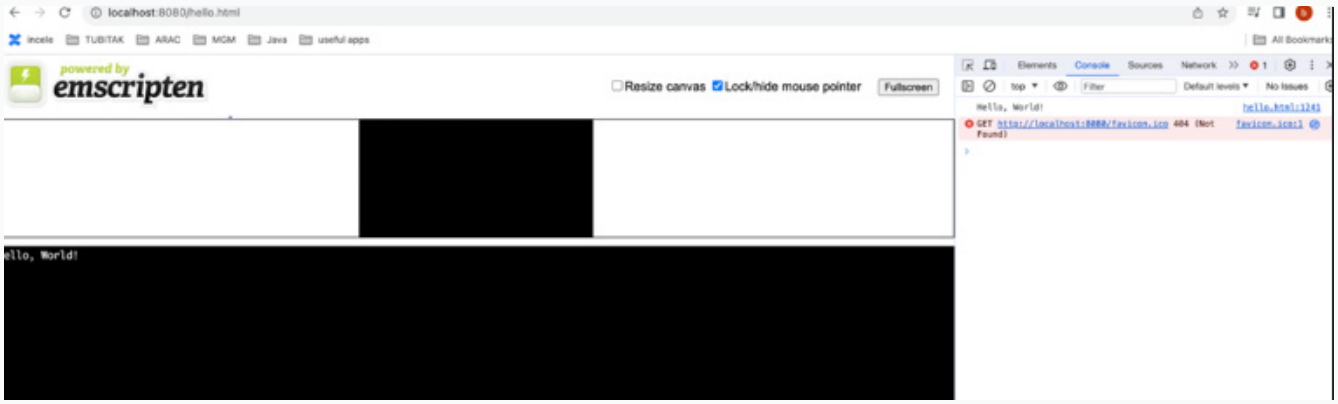
Şekil 11. WASM Çıktısının HTML İçinde Kullanımı

C++

Benzer şekilde ama bu sefer Rust kullanmayıp, direkt sadece C++ ile alert() atma denendi. “em++ hello.cpp -o hello.html” komutu ile Emscripten aracına Şekil 12'deki C++ kodu derletildi ve Şekil 13’teki gibi bir çıktı elde edildi.

```cpp
int main() {
  std::cout << "Hello, World!" << std::endl;
  return 0;
}
```

Şekil 12. C++ ile "Hello World!" Kodu



Şekil 13. C++ Kodunun Emscripten Çıktısı

100. Fibonacci Sayısı Performans Testi

Bu denemede ise performans kazancı test edildi. Kod iterative (yinelemeli) şekilde yazıldığında Rust'ın, JavaScript'ten yaklaşık 110 kat daha hızlı olduğu tespit edildi (bkz. Şekil 14). Lakin, kod recursive (öz yinelemeli) şeklinde yazıldığında ise JavaScript, Rust'ı az farkla yenmeyi başardı (Chromium JIT derleyicisinin gücü). Kullanılan kodlar için Şekil 15 ve 16 incelenebilir.

```
bilalozlu@administrator-MacBook-Pro webasm % ./main
result is 633825300114114700748351602688
Elapsed: 26.75µs

bilalozludadministrator-MacBook-Pro webasm % node fibo.js
result is 6.338253001141147e+29
Elapsed: 3ms
```

Şekil 14. JS ve Rust Kodunun Hızını Gösteren Terminal Çıktısı

```
function fib(n: number) {
  let a: number = 0;
  let b: number = 1;
  for (let i: number = 0; i < n; i++) {
    a = b;
    b = a + b;
  }
  return a;
}

const startTime: number = new Date().getTime();

const result: number = fib(100);
console.log('result is', result);

const endTime: number = new Date().getTime();
const elapsed: number = endTime - startTime;
console.log('Elapsed:', elapsed, 'ms');
```

Şekil 15. JS ile Fibonacci Kodu

```
fn fib(n: i128) → i128 {
  let mut a: i128 = 0;
  let mut b: i128 = 1;
  for _ in 0..n {
      a = b;
      b = a + b;
  }
  return a
}

fn main() {
  use std::time::Instant;

  let now = Instant::now();

  let result = fib(100);
  println!("result is {}", result);

  let elapsed = now.elapsed();
  println!("Elapsed: {:.2?}", elapsed);
}
```

Şekil 16. Rust Ile Fibonacci Kodu

Sonuç ve Öneriler

WebAssembly ve Rust, web geliştirme dünyasında son yıllarda oldukça popüler hale gelen ve doğru kullanıldığında web geliştirmenin geleceği için büyük bir potansiyele sahip olan iki yenilikçi teknolojidir. Birlikte kullanıldıklarında, daha hızlı, daha güvenli ve daha ölçeklenebilir web uygulamaları oluşturmak için güçlü bir platform sunarlar. Bu çalışmada, WebAssembly ve Rust'ın temelleri, birlikte nasıl kullanılacağı ve web geliştirmeye sundukları avantajlar ayrıntılı olarak incelendi.

Ancak WebAssembly ve Rust'ın hala nispeten yeni teknolojiler olduğunu ve bazı tarayıcılarda veya platformlarda beklenmedik davranışlarla karşılaşılabileceğini unutmamak önemlidir. Bu nedenle bu teknolojileri kullanırken dikkatli olmak ve yeni gelişmelerden haberdar olmak önemlidir. WebAssembly ve Rust ile geliştirmeye başlamak ve gelişmelerden haberdar olmak isteniyorsa, yapılabileceklerden bazıları şöyledir;

- **WebAssembly'yi öğrenin:** WebAssembly'yi öğrenmek için birçok kaynak mevcuttur. MDN Web Docs'taki WebAssembly sayfası iyi bir başlangıç noktasıdır.
- **Rust'ı öğrenin:** Rust'ı öğrenmek için birçok kaynak mevcuttur. Resmi Rust web sitesi iyi bir başlangıç noktasıdır.
- **Mevcut projeleri keşfedin:** Rust ve WebAssembly ile oluşturulmuş birçok mevcut proje vardır. Bu projeleri keşfetmek ve ilham almak için harika bir yoldur. Mevcut projeleri listeleyen kaynaklardan bazıları:
  - **WASM:** madewithwebassembly.com/
  - **Rust:** github.com/omarabid/rust-companies

WebAssembly veya Rust kullanmaya başlamak isteniyorsa, araştırma yapılması ve bu teknolojilerin geliştiricileriniz ve kullanıcılarınız için uygun olup olmadığının -yani yeni ekosistemin getirdiği zorlukların alınacak performansı karşıladığını- belirlenmesi tavsiye edilir. Yeni projelerde -Figma'nın da yaptığı gibi- kullanıcı arayüzünü daha standart React, Vue gibi çerçevelerle yapıp asıl performans gerektirecek kısımlar WASM ile yapılabilir. Bu şekilde alınan risk en aza indirilirken en fazla kazanç sağlanabilir.

Kaynakça

1. Rust Programming Language, URL: https://rust-lang.org (Haziran, 19, 2024).
2. Docs.rs, URL: https://docs.rs/ (Haziran, 19, 2024).
3. GitHub, URL: https://github.com/muratcabuk/makaleler/blob/master/webassembly/webassembly.md (Haziran, 19, 2024).
4. GitHub, URL: https://github.com/omarabid/rust-companies (Haziran, 19, 2024).
5. TechTarget, URL: https://techtarget.com/searchitoperations/definition/WebAssembly (Haziran, 19, 2024).
6. Github, URL: https://rustwasm.github.io/docs/book/why-rust-and-webassembly.html (Haziran, 19, 2024).
7. Medium, URL: https://paulburkart.medium.com/pros-and-cons-of-the-rust-programming-language-50acbb558bec (Haziran, 19, 2024).
8. Thurrott, URL: https://thurrott.com/windows/282471/microsoft-is-rewriting-parts-of-the-windows-kernel-in-rust (Haziran, 19, 2024).
9. Career Karma, URL: https://careerkarma.com/blog/companies-that-use-rust/ (Haziran, 19, 2024)
10. web.dev, URL: https://web.dev/articles/ps-on-the-web (Haziran, 19, 2024).
11. web.dev, URL: https://web.dev/case-studies/earth-webassembly (Haziran, 19, 2024).
12. Figma, URL: https://figma.com/blog/webassembly-cut-figmas-load-time-by-3x/ (Haziran, 19, 2024).
13. Figma, URL: https://figma.com/blog/rust-in-production-at-figma/ (Haziran, 19, 2024).
14. InfoWorld, URL: https://infoworld.com/article/3254574/microsofts-blazor-project-runs-net-in-the-browser.html (Haziran, 19, 2024).
15. InfoWorld, URL: https://infoworld.com/article/3269478/how-to-run-net-apps-in-the-browser-with-blazor.html (Haziran, 19, 2024).
16. Wikipedia, URL: https://en.wikipedia.org/wiki/Fabrice_Bellard (Haziran, 19, 2024).
17. Unity, URL: https://blog.unity.com/engine-platform/webassembly-is-here (Haziran, 19, 2024).
18. Wonder, URL: https://theimmersiveweb.com/blog (Haziran, 19, 2024).

T.C. SANAYİ VE
TEKNOLOJİ BAKANLIĞI

#MİLLİ TEKNOLOJİ HAMLESİ

TÜBİTAK

İşçi Blokları Mahallesi Muhsin Yazıcıoğlu Caddesi No:51/C 06530 Çankaya/ANKARA

+90 (312) 289 92 22 - yte.bilgi@tubitak.gov.tr

TÜBİTAK - BİLGEM Yazılım Teknolojileri Araştırma Enstitüsü (YTE)