POGLAVLJE 1

Programeri sistema mogu imati lepe stvari

U određenim kontekstima – na primer, kontekst na koji Rust cilja – je 10x ili čak 2x brži od konkurencije. Brzina odlučuje o sudbini na tržištu, kao što odlučuje na tržištu hardvera.
Greydon Hoare

Svi računari su sada paralelni...
*Paralelno programiranje **jeste** programiranje.*
Michael McCool i saradnici, *Structured Parallel Programming*

Greška u paraseru TrueTypa koriščena od američke države koristi za sveopšti nadzor; sav softver je bezbednosno osetljiv.
Andy Wingo

Odlučili smo da počnemo ovu knjigu sa tri navedena citata. Ali počnimo sa misterijom. Šta radi sledeći C program?

```
int main(int argc, char **argv) {
  unsigned long a[1];
  a[3] = 0x7ffff7b36cebUL;
  return 0;
}
```

Na Džimovom laptopu jutros je ovaj program ispisao:

```
undef: Error: .netrc file is readable by others.
undef: Remove password or make file unreadable by others.
```

Onda se srušio. Ako ga isprobate na svojoj mašini, možda će uraditi nešto drugo. Šta se ovde dešava?

Program je pogrešan. Niz `a` je dugačak samo jedan element, tako da je korišćenje `a[3]`, prema standardu programskog jezika C, *nedefinisano ponašanje*:

> Ponašanje pri korišćenju neprenosive ili pogrešne programske konstrukcije ili pogrešnih podataka, za koje međunarodni standard ne nameće nikakve zahteve

Nedefinisano ponašanje nema samo nepredvidiv rezultat: standard eksplicitno dozvoljava programu da uradi *bilo šta*. U našem slučaju, čuvanje ove određene vrednosti u četvrtom elementu ovog određenog niza narušava stek poziva funkcije pri vraćanju iz `main` funkcije, umesto elegantnog izlaska iz programa kako bi trebalo, skače usred koda iz standardne C

biblioteke za preuzimanje lozinke iz datoteke u korisničkom kućnom direktorijumu. To nije dobro.

C i C++ imaju stotine pravila za izbegavanje nedefinisanog ponašanja. Oni su uglavnom zdravorazumski: ne pristupajte memoriji kojoj ne bi trebalo, ne koristite nedozvoljene aritmetičke operacija, ne delite sa nulom itd. Ali kompajler ne sprovodi ova pravila; nema obavezu da otkriva čak ni eklatantne prekršaje. Zaista, prethodni program se kompajlira bez grešaka ili upozorenja. Odgovornost za izbegavanje nedefinisanog ponašanja u potpunosti pada na vas, programera.

Empirijski gledano, mi programeri nemamo sjajne rezultate u ovom pogledu. Dok je bio student na Univerzitetu Juta, istraživač Peng Li je modifikovao C i C++ kompajlere da bi programi koje je prevodio izvestili da li su izvršili određene oblike nedefinisanog ponašanja. Otkrio je da skoro svi programi to čine, uključujući i one iz dobro cenjenih projekata koji svoj kôd drže po visokim standardima. Pretpostavka da možete da izbegnete nedefinisano ponašanje u C i C++ je kao pretpostavka da možete pobediti u partiji šaha samo zato što znate pravila.

Povremena čudna poruka ili rušenje mogu biti problem kvaliteta, ali nenamerno nedefinisano ponašanje je glavni uzrok bezbednosnih propusta kada je Morrisov crv 1988. god. iskoristio varijaciju tehnike prikazane ranije, za širenje sa jednog računara na drugi na ranom Internetu.

Dakle, C i C++ dovode programere u nezgodnu poziciju: ti jezici su industrijski standardi za programiranje sistema, ali zahtevi koje postavljaju programerima samo garantuju stalna rušenja i bezbednosne probleme. Odgovor na misteriju postavlja veće pitanje: zar ne možemo bolje?

Rust peuzima vaš teret na sebe

Naš odgovor je uokviren sa naša tri uvodna citata. Treći citat se odnosi na izveštaje da je Stuxnet, kompjuterski crv otkriven u provali u industrijsku kontrolnu opremu 2010. godine, preuzeo je kontrolu nad računarima žrtava koristeći, između mnogih drugu tehniku, nedefinisano ponašanje u kodu koji je parsirao TrueType fontove ugrađene u dokumente za obradu teksta. Možemo biti sigurni da autori tog koda nisu očekivali da će se koristiti na ovaj način, što ilustruje da nisu samo operativni sistemi i serveri ti koji treba da brinu o bezbednosti: svaki softver koji može da obrađuje podatke iz nepouzdanog izvora može biti meta zloupotrebe.

Rust jezik vam daje jednostavno obećanje: ako vaš program prođe provere kompajlera, nema nedefinisanog ponašanja. Viseći pokazivači, double free greške i pozivanja null pokazivača se hvataju u vreme kompajliranja. Reference niza su obezbeđene mešavinom provera vremena kompajliranja i vremena izvođenja, tako da nema prekoračenja bafera: Rust ekvivalent našeg nesrećnog C programa bezbedno javlja poruku o grešci.

Dalje, Rust ima za cilj da bude *bezbedan* i *prijatan za korišćenje*. Da bi dao jače garancije o ponašanju vašeg programa, Rust nameće više ograničenja vašem kodu nego C i C++, a ova ograničenja zahtevaju praksu i iskustvo da biste se navikli. Ali jezik u celini je fleksibilan i izražajan. O tome svedoči obim koda napisanog u Rustu i opseg oblasti primene na koje se primenjuje.

Po našem iskustvu, mogućnost da raspolažemo jezikom kome verujemo da hvata više grešaka podstiče nas da pokušamo sa ambicioznijim projektima. Izmene velikih, složenih programa su manje rizične kada znate da su rešeni problemi upravljanja memorijom i validnosti pokazivača. A otklanjanje grešaka je mnogo jednostavnije kada potencijalne posledice greške ne uključuju oštećenje nepovezanih delova programa.

Naravno, još uvek postoji mnogo grešaka koje Rust ne može da otkrije. Ali u praksi, uklanjanje nedefinisanog ponašanja bitno menja karakter razvoja na bolje.

Paralelno programiranje je ukroćeno

Istovremenost (eng. concurrency) je notorno teško koristiti pravilno u C i C++. Programeri se obično okreću istovremenosti samo kada se pokaže da jednonitni kôd nije u stanju da postigne performanse koje su im potrebne. Ali drugi uvodni citat tvrdi da je paralelizam previše važan za moderne mašine da bismo ga tretirali kao krajnju instancu.

Kao što se ispostavilo, ista ograničenja koja obezbeđuju bezbednost memorije u Rustu takođe obezbeđuju da Rust programi nemaju trku podataka. Možete slobodno da delite podatke između niti, sve dok se ne menjaju. Podacima koji se menjaju može se pristupiti samo pomoću primitiva za sinhronizaciju. Dostupni su svi tradicionalni alati za konkurentnost: muteksi, promenljive uslova, kanali, atomika itd. Rust jednostavno proverava da li ih pravilno koristite.

Ovo čini Rust odličnim jezikom za iskorišćenje sposobnosti savremenih mašina sa više jezgara. Rust ekosistem nudi biblioteke koje prevazilaze uobičajene paralelne primitive i pomažu vam da ravnomerno rasporedite složena opterećenja na skupove procesora, koristite mehanizme sinhronizacije bez zaključavanja kao što je Read-Copy-Update i još mnogo toga.

I pored svega Rust je i dalje brz

Ovo je, konačno, naš prvi uvodni citat. Rust deli ambicije koje Bjarne Stroustrup artikuliše za C++ u svom radu „Abstraction and the C++ Machine Model“:

> Generalno, implementacije C++-a poštuju princip nultih troškova: ono što ne koristite, ne plaćate. I dalje: Ono što koristite, ne može biti bolji kôd.

Programiranje sistema se često bavi guranjem mašine do njenih granica. Za video igre, cela mašina treba da bude posvećena stvaranju najboljeg iskustva za igrača. Za veb čitače, efikasnost čitača postavlja gornju granicu onoga što autori sadržaja mogu da urade. U okviru inherentnih ograničenja mašine, što je više moguće memorije i pažnje procesora mora biti

prepušteno samom sadržaju. Isti princip važi i za operativne sisteme: kernel treba da učini resurse mašine dostupnim korisničkim programima, a ne da ih sam troši.

Ali kada kažemo da je Rust „brz“, šta to zaista znači? Može se napisati spor kôd na bilo kom jeziku opšte namene. Preciznije bi bilo reći da, ako ste spremni da investirate u dizajniranje vašeg programa kako biste na najbolji način iskoristili mogućnosti mašine ispod, Rust vas podržava u tom nastojanju. Jezik je dizajniran sa efikasnim podrazumevanim podešavanjima i daje vam mogućnost da kontrolišete kako se memorija koristi i kako se troši pažnja procesora.

Rust čini saradnju jednostavnijom

Sakrili smo četvrti citat u naslov ovog poglavlja: „Programeri sistema mogu da imaju lepe stvari“. Ovo se odnosi na Rustovu podršku za deljenje koda i ponovnu upotrebu.

Rustov menadžer paketa i alatka za pravljenje, Cargo, olakšava korišćenje biblioteka koje su objavili drugi na Rustovom javnom spremištu paketa, veb sajtu crates.io. Jednostavno dodate naziv biblioteke i potreban broj verzije u datoteku, a Cargo se brine o preuzimanju biblioteke, zajedno sa svim drugim bibliotekama koje koristi, i povezivanju čitave serije. Možete zamisliti Cargo kao Rustov odgovor na NPM ili RubyGems, sa naglaskom na zdravo upravljanje verzijama i ponovljive verzije. Postoje popularne Rust biblioteke koje pružaju sve, od standardne serijalizacije do HTTP klijenata i servera i modernih grafičkih API-ja.

Idući dalje, sam jezik je takođe dizajniran da podrži saradnju: Rustove osobine i generika omogućavaju vam da kreirate biblioteke sa fleksibilnim interfejsima tako da mogu da služe u mnogo različitih konteksta. A Rustova standardna biblioteka pruža osnovni skup osnovnih tipova koji uspostavljaju zajedničke konvencije za uobičajene slučajeve, čineći različite biblioteke lakšim za zajedničko korišćenje.

Sledeće poglavlje ima za cilj da konkretizuje široke tvrdnje koje smo izneli u ovom poglavlju, uz obilazak nekoliko malih Rust programa koji pokazuju snagu jezika.