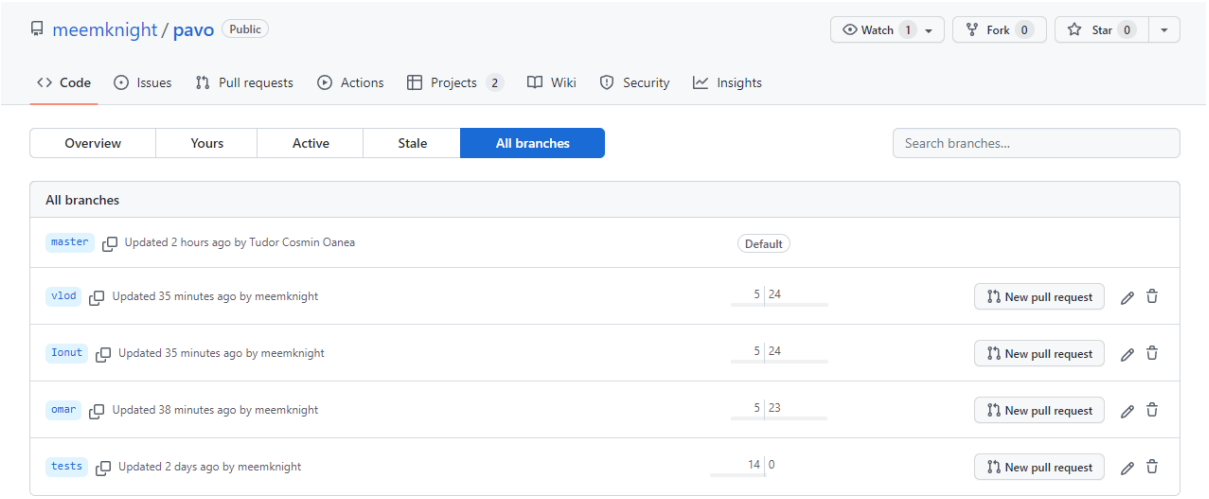
User Stories

Design / Arhitectura:

Problem: Debuggerul trebuie sa mearga si pe windows si pe linux, si sa aiba 2 implementari de interfete: consola + interata grafica.

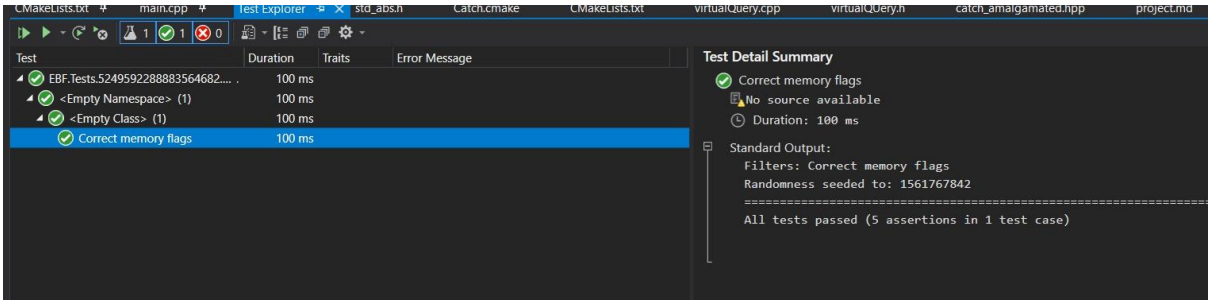
Solutie: Avem o clasa Debugger care are un api unde putem trimite comenzi. Clasa debugger are cateva functii de genu read_memory set_breakpoint get_pc etc care au fiecare 2 implementari: una pentru windows una pentru linux. Functia handle_command este cross platform(folosind functiile specificate mai sus) si asteapta o comanda (exemplu seteaza sau citeste un registru) functia asta o chemam din cli sau din gui respectiv.

Source control (branch creation, merge/rebase, minim 10 commits):



Teste automate

Am creat 5 teste automate pentru a verifica parsarea corecta a memoriei. Intregrarea in aplicatie am facut-o cu ajutorul librariei Catch2, rezultatul testelor este afisat in Microsoft Visual Studio dupa cum urmeaza:



Fiecare test asigura parsarea si cititrea corecta a memoriei in fiecare caz specific tipului de memorie alocat in functie de urmatoarele flag-uri: Read, Execute, Read_Write,

Execute_Read si Execute_Read_Write. Astfel, periodic, inainte de rularea codului principal, am folosit aceste teste cu rolul de a garanta urmatoarele executii ale programului in regim optim.

Bug reporting

La hex editor am întâlnit un bug atunci când adresa de la care trebuia sa inceapa afisarea memorie se apropia de limita teoretica a memoriei - 2^64 - 1 (cel mai mare întreg ce poate fi scris pe 64 de biti fara semn), ceea ce conducea la un overflow când peste adresa de bază era adaugata dimensiunea zonei de memorie pe care doream sa o afisam.

Pentru a rezolva aceasta problema am limitat valoarea maximă a adresei de start astfel incat sa prevenim un overflow, dar sa fie în continuare posibil sa afisam corect starea ultimilor “bytes” de memorie. Practic, la acea adresa ar fi oricum imposibil sa găsim memorie alocată din moment ce nici un computer nu dispune de suficientă memorie RAM incat sa ocupe întreg spațiul de adresare oferit de sistemele pe 64 de biti. În plus, procesoarele disponibile în prezent folosesc doar 48 de biți dintr-o adresa din moment ce aceasta dimensiune este suficientă pentru a adresa 256 TB de memorie.
needs a link to bug issue on github / commit in which it was fixed

Folosirea unui build tool

Am scris proiectul in Microsoft Visual Studio si am folosit CMake.

Refactoring

Avand in vedere faptul ca numeroase situatii in care am scris cod mult prea specific si greu reutilizabil au aparut in procesul dezvoltarii aplicatiei noastre, devine limpede evidentiata nevoia de a aloca timp pentru procedeul de refactoring. Un exemplu de cod care a fost transformat in unul reutilizabil si usor de citit este aici.

Design patterns

Am folosit un singleton cu implementare neuzuala pentru a incarca o singura data informatiile necesare despre CPU.