Съдържание

Увод

Модерните процесори разчитат на два основни принципа, за да предоставят потребителите си множество програми, които да работят едновременно:

- Context switching (Смяна на контекста за изпълнение);
- Multiprocessing (Мултипроцесинг).

Context switching е способността на един процесор да спре изпълнението си над една програма и да изпълни друга за известно време. Това също помага на много нишкови процеси да останат активни, докато извършват работа в заден план.

Multiprocessing е присъствието на повече от едно ядро в централния процесор за паралелно изпълнение на програми. Този метод принципно се комбинира с Multithreading и Context switching, за да предостави максимална гъвкавост и позволява истински “multi-tasking”.

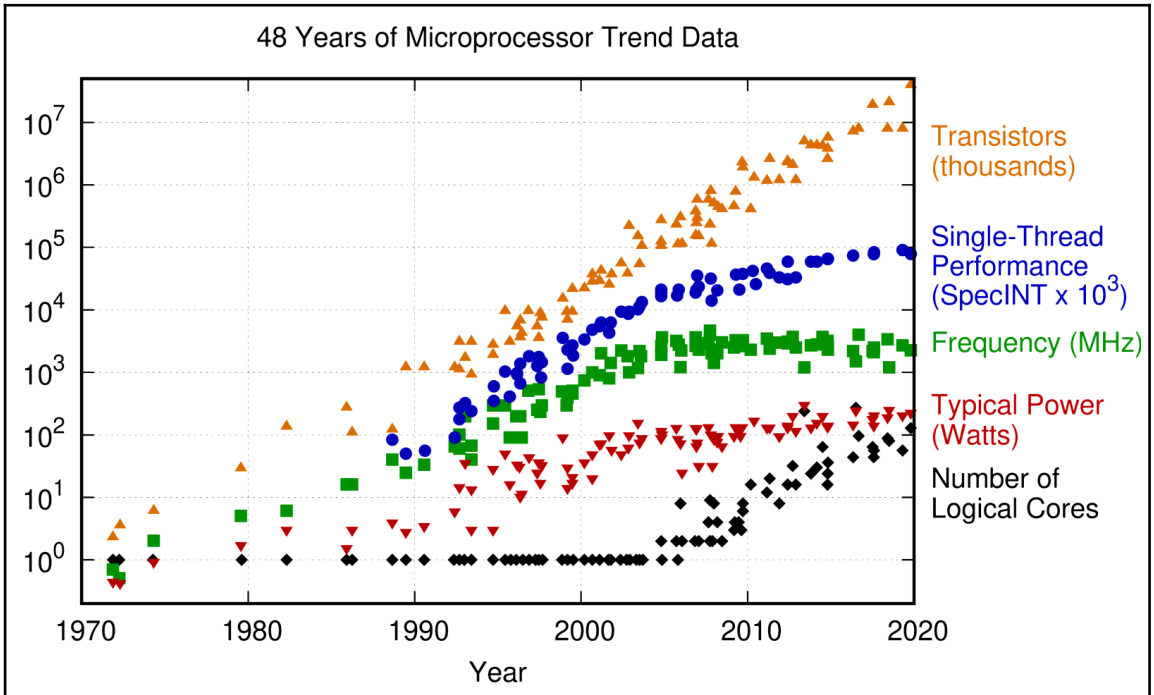
Process Affinity е терминът даден на колекцията от ядра или нишки, където е позволено един процес да бъде изпълнен. Това дефакто определя в кое ядро ще се изпълни програмата и с кои други програми ще “споделя” ресурси. По подразбиране всички модерни операционни системи за x86 и AMD64 позволяват на всички програми да работят при всяко ядро/нишка. Също така играе роля приоритета на процеса, тъй като процесите с по-висок приоритет получават повече време за изпълнение в сравнение с други процеси с по-нисък приоритет.

Докато тези настройки могат да бъдат променени сравнително лесно за един процес, в наличието на стотици процеси във всяка дадена система прави ръчното настройване на тези настройки прекалено трудно и бавно, което прави автоматизацията на този процес софтуерно инженерно предизвикателство.

В дипломната работа са описани основните действащи части и предизвикателства на такава система за автоматизация, какви допълнителни настройки и системи за качество на живот са предвидени и какво е мястото на решението в пазара.

Нуждата от ръчен контрол и предимства над автоматични режими на операционни системи

Първите многоядрени процесори станаха популярни сред ентусиасти около 2004-та година, като през следващите 16 години броят на техните ядра се увеличи многократно.



Фиг. 1 - Брой транзистори, SpecINT производителност, мощност и **брой логически ядра** на процесори от 1970 до 2020 г. https://github.com/karlrupp/microprocessor-trend-data

Заедно с увеличаването на броя ядра и нишки, изискванията на потребителските софтуерни решения също се увеличиха, но операционните системи продължават да третират всички програми в една система по същият начин, със същия приоритет. Докато това е добро решение за по-голямата част от потребители, за тъй наречените “power user”и, нуждата от по-голям контрол стана очевидна.

Добър пример за това са проблемите с регресия в тестваната производителност на процесора от AMD, модел Threadripper 2990WX под Windows операционни системи през края на 2018-та и началото на 2019-та година, поради различната топография на процесорната архитектура, по-скъпия и по-бързия (на теория) 2990WX работеше по-бавно от 2950X.[9][10]

Проблема беше отстранен с тестване и опити над “Process Affinity” и “Process Priority” на тестваните програми. В днешно време, различните ядра на един процесор могат да имат имат различно време на достъп до дадена банка от памет и различни програми реагират по различни начини на тези закъснения при достъпа до паметта. Докато в този случай специфичните проблеми и приложеното решение навлизат в много повече детайли, и те са извън обхвата на тази дипломна работа, нуждата от по-голям контрол е очевидна.[9][11]

Освен тези проблеми с производителността, някои Intel процесори също страдат от **PortSmash** критична уязвимост, позволяваща на процесите споделящи едно и също ядро с включен HyperThreading. Програми като PACT могат да предотвратят този вектор на нападение като изолират уязвими и критични процеси в други ядра/нишки от останалите процеси в системата. [12][13][14]

Анализ на съществуващи решения, техните предимства и недостатъци

Това, че има съществуващи решения на този проблем, не означава, че няма нужда или място в пазара от други алтернативни решения. Всички решения изброени тук покриват части от моите лични нужди или изисквания, но не и всички.

Най-вече всички готови решения са със затворен код, което означава, че най-вероятно има потребители, чиито нужди не са покрити от тези готови решения.

Някои от тези, вече съществуващи решения са:

- Софтуерни:
    - BitSum Process Lasso (2011);
    - CPUCores (2015);

- Вградени:
    - Windows Task Manager;
    - htop за Linux операционни системи;
    - Bash / Powershell скриптове.

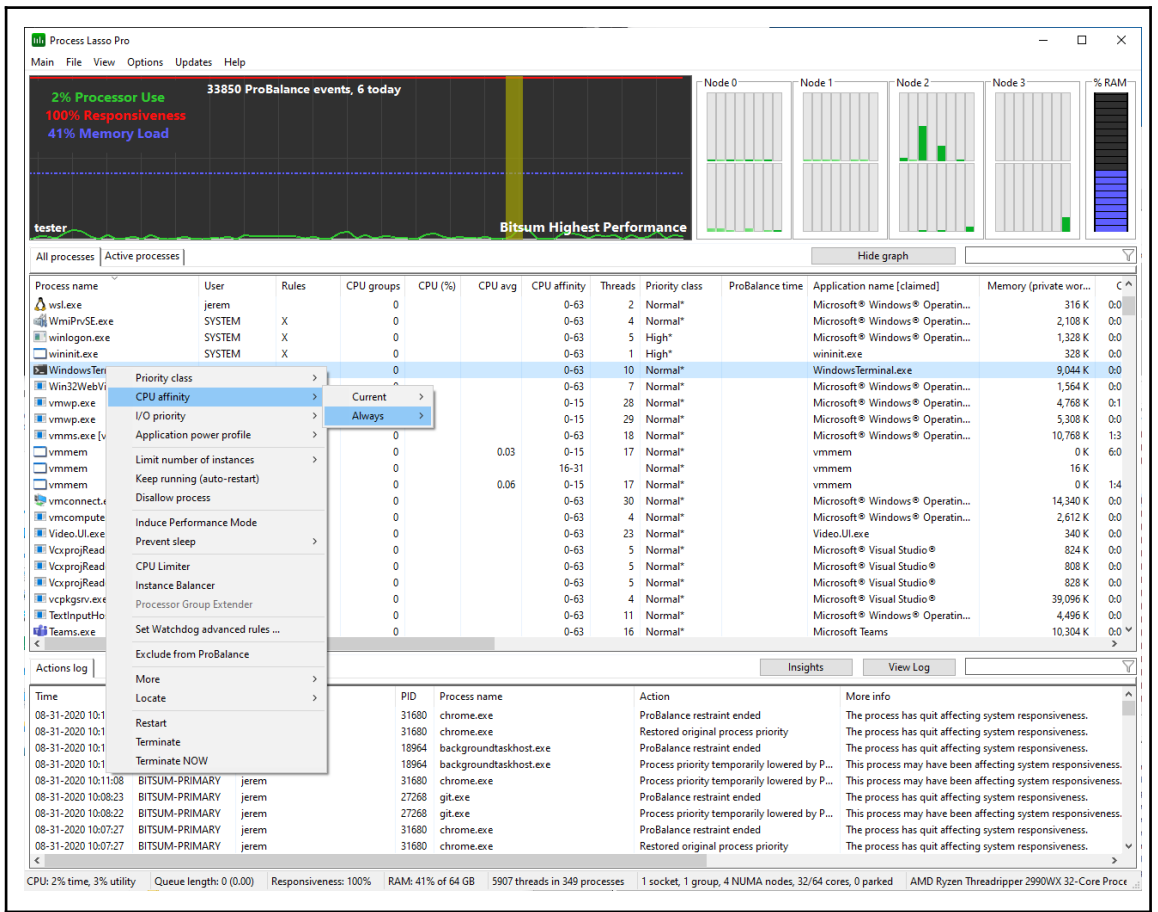
Предимствата и недостатъците на тези решения варират.
За софтуерните решения, предназначените им употреби са напълно противоположни едни от други.
Вградените решения нямат достъпни начини за автоматизация от един нормален потребител.

Изброените, решения имат следните предимства и недостатъци:

BitSum Process Lasso (2011)

Process Lasso от BitSum е софтуерно решение за Windows операционни системи. Предлага множество алгоритми и под-системи за управление на приоритет на процесите и тяхното разпределение по ядрата/нишките за даден процесор.

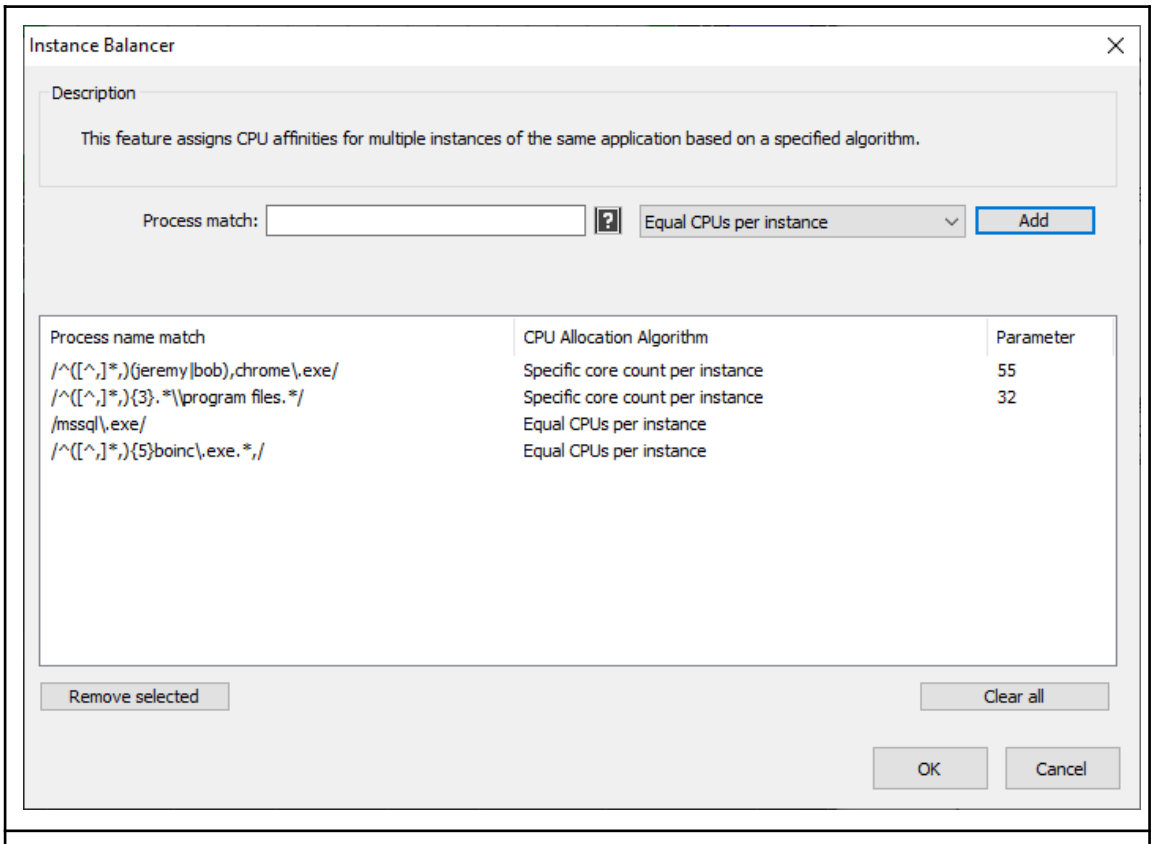
Предназначен е най-вече за системни администратори на Windows базирани сървъри, предлага разновидни автоматизирани операции за системна администрация като рестартиране на процеси и лимитиране на броя процеси от даден клас.



Фиг. 2 - Главния прозорец на графичната среда на Process Lasso. (https://bitsum.com/)

Process Lasso предлага следните предимства:

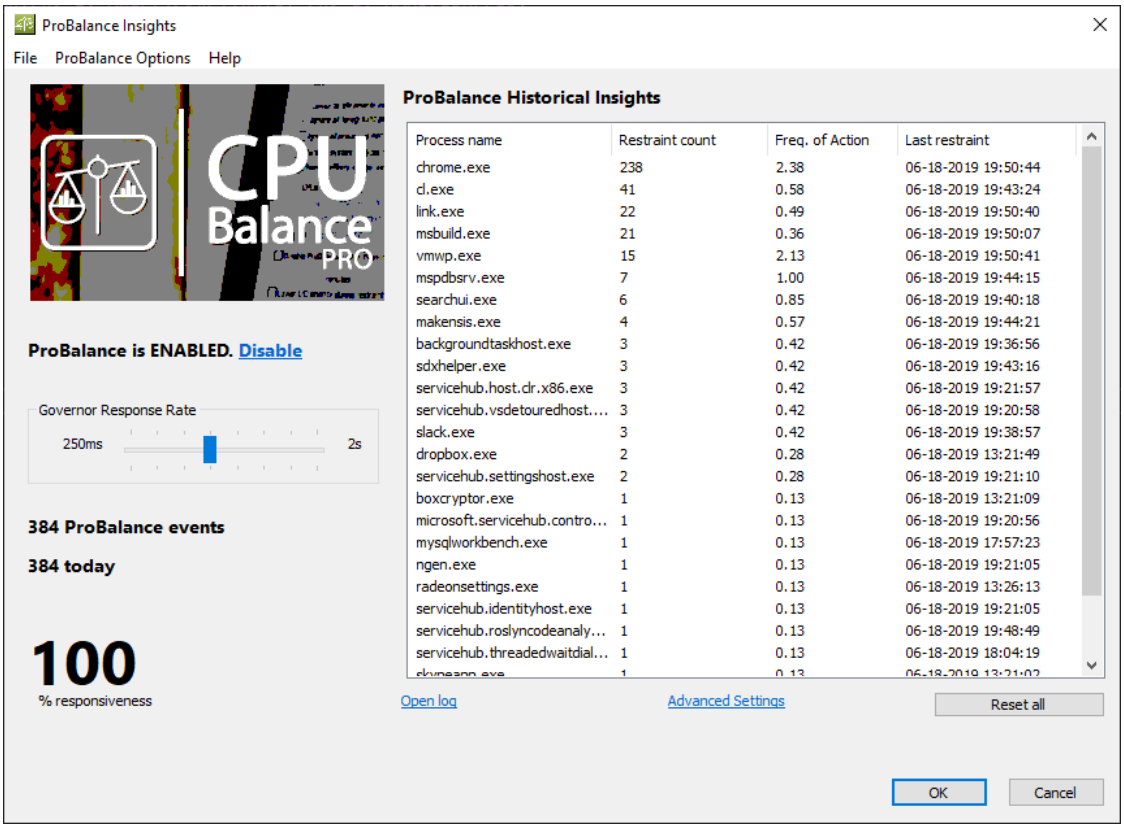
- **По-подробни конфигурации** - Process Lasso е предназначен за максимален контрол над операциите на един сървър, следователно конфигурациите имат по-подробни детайли, с които да се ограничи операцията на даден процес.
- **Филтърни конфигурации** - Софтуера може да прилага конфигурации, базирани на RegEx филтри *(Фиг. 3)*
- **Балансирани изпълнения на процеси** - Ограничава броя процеси и тяхното време на изпълнение в системата.
- **Enterprise поддръжка** - Изключително важен фактор за бизнеси е официалната поддръжка на софтуера.



Фиг. 3 - Process Lasso може да прилага конфигурации, базирани на сложни филтри. (https://bitsum.com/)

Process Lasso има следните недостатъци:

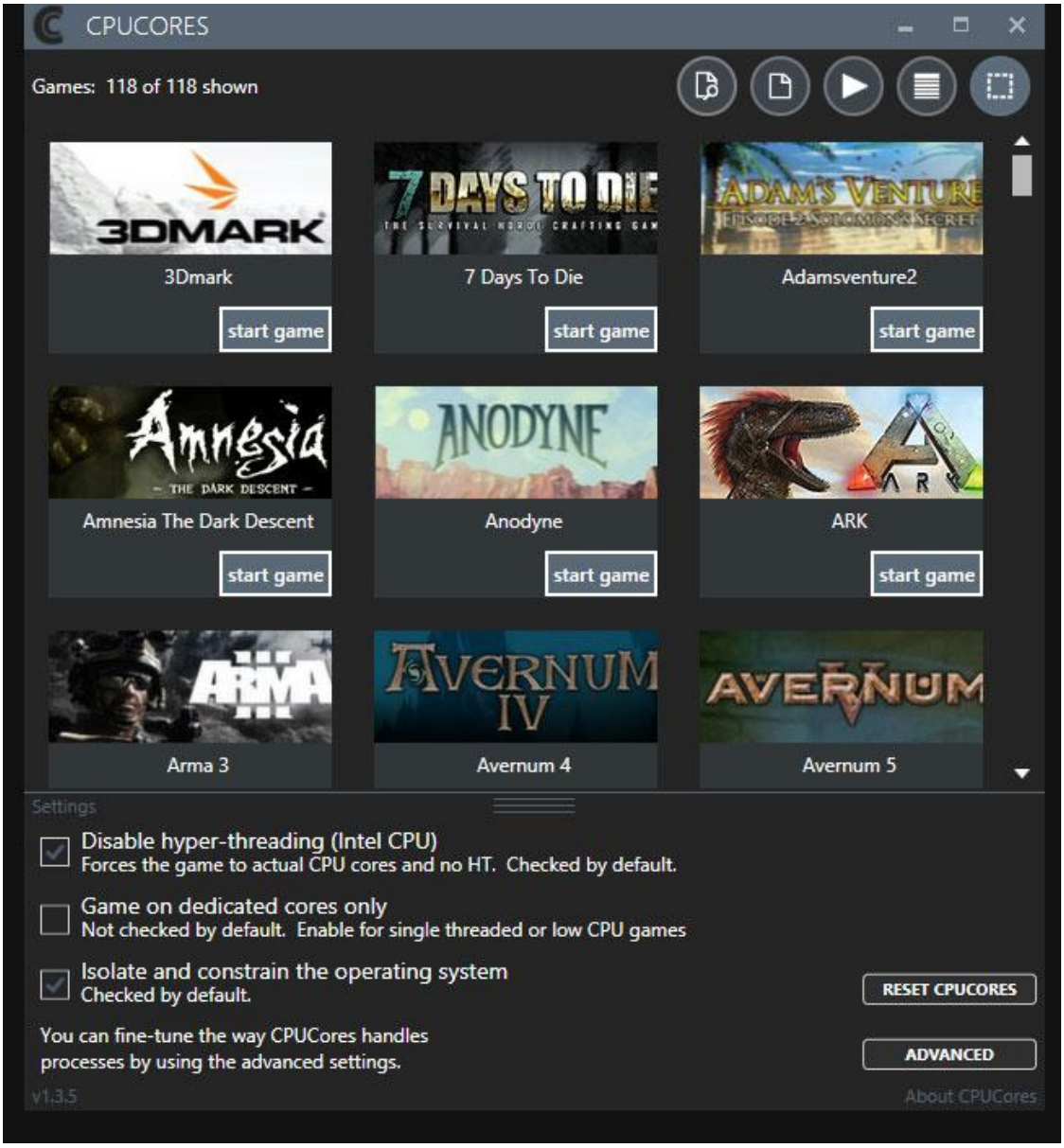
- **Платен софтуер, изисква лиценз** - Не е подходящ за повечето потребители с нисък или нулев бюджет.
- **Само за Windows операционни системи** - Докато Windows е навсякъде в десктоп пазара, сървърния пазар е почти напълно превзет от Linux базирани операционни системи.
- **Затворен код** - Намалява гъвкавостта на софтуера като не позволява на потребителя да промени части от кода, за да го адаптира на техните нужди.
- **Предназначен е за мощни системи** - Process Lasso е предназначен да балансира системи с много ядра/нишки и процеси, не е предназначен за нормални потребители.



Фиг. 4 - ProBalance позволява балансирано време за достъп до процеси. (https://bitsum.com/)

CPUCores (2015)

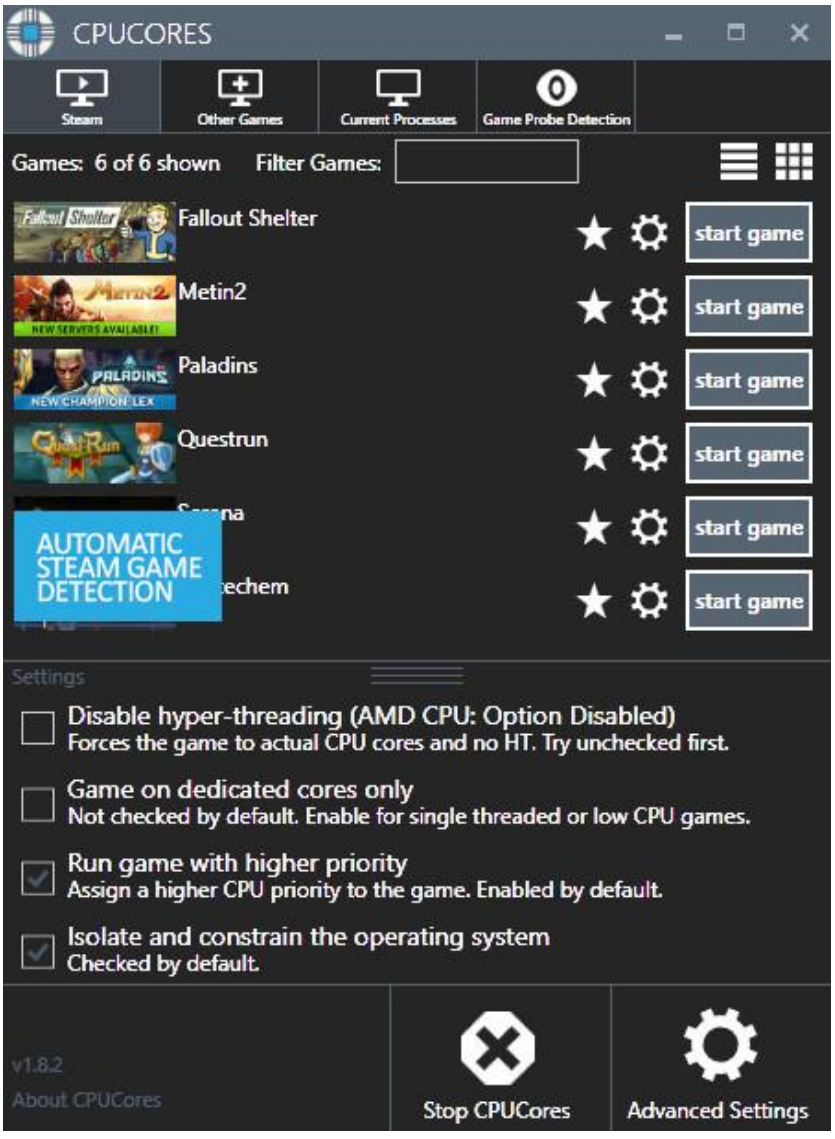
CPUCores е софтуерно решение предлагано от Tim Sullivan. Основно е предназначен за потребители, които играят видео игри и следователно целия софтуер е ориентиран да бъде лесен за употреба и предлага по-малко настройки.



Фиг. 5 - Главния прозорец на графичната среда на CPUCores. (http://cpucores.com/)

CPUCores предлага следните предимства:

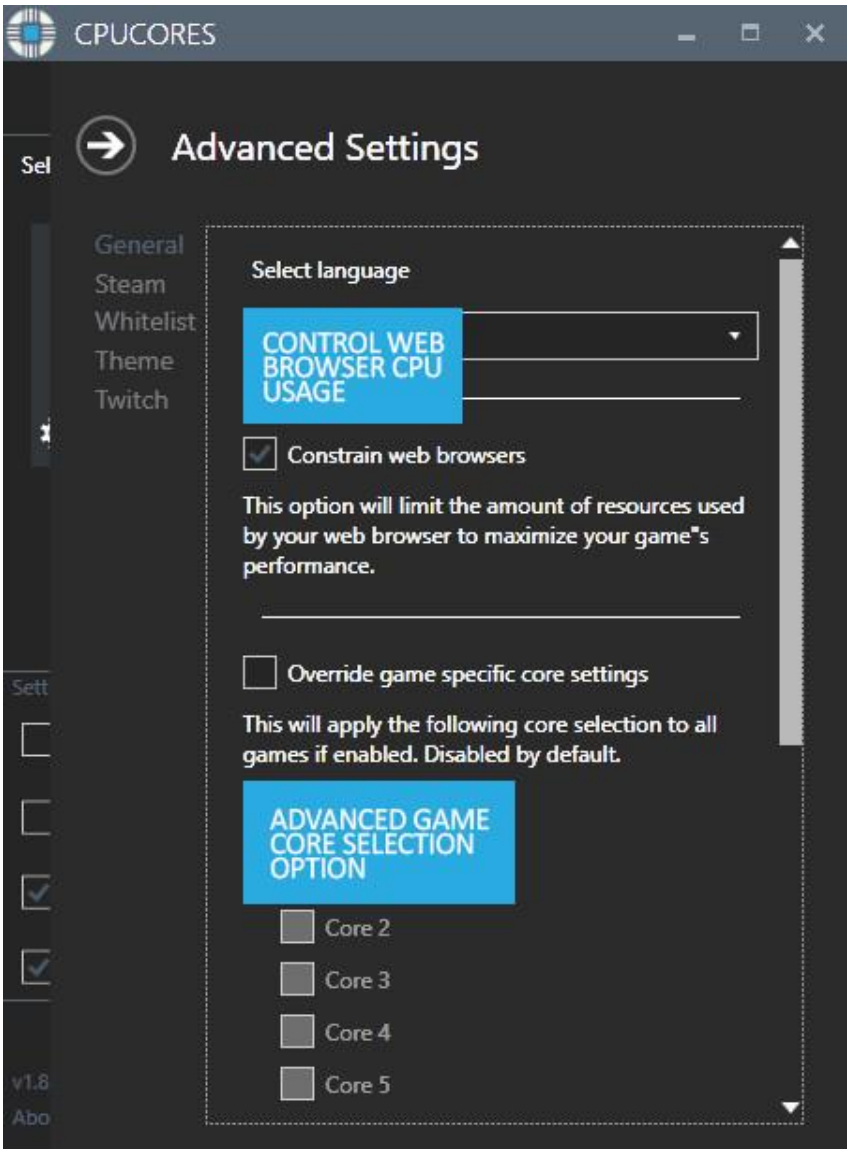
- **Филтърни конфигурации**
- **Автоматично отчитане на видео игри** - Докато това е потребно за потребители, които играят видео игри, не помага особено за други. *(Фиг. 6)*
- **Оптимизации на операционната система** - Детайли за как работи това не са дадени.



Фиг. 6 - Главния прозорец на графичната среда на CPUCores. (http://cpucores.com/)

Process Lasso има следните недостатъци:

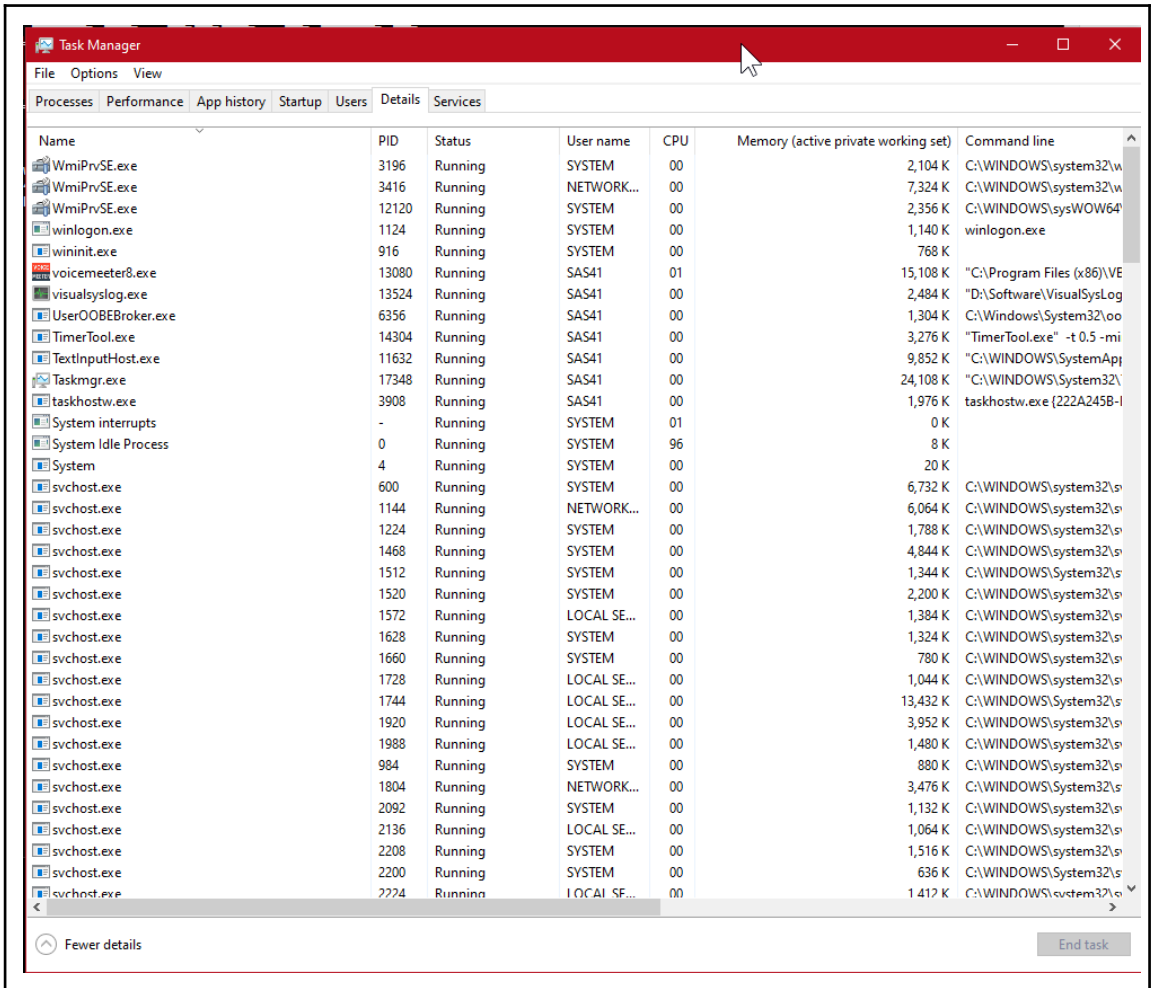
- **Платен софтуер**
- **Затворен код**
- **Само за Windows операционни системи**
- **Недокументирани оптимизации** - Тези промени за оптимизации не са документирани и може да водят до нежелани странични ефекти.
- **Предназначен е да се използва за видео игри**



Фиг. 7 - Конфигурационния прозорец на CPUCores. (http://cpucores.com/)

Windows Task Manager и HTOP за Linux базирани операционни системи

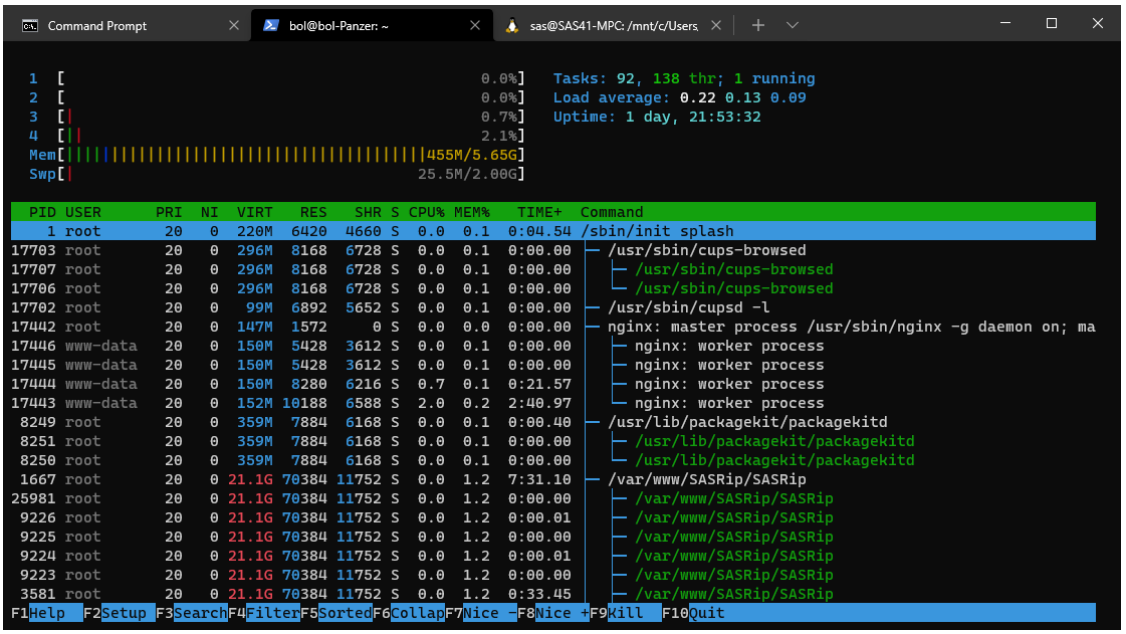
Windows Task Manager и HTOP за Linux базирани операционни системи имат за цел да информират потребителя за използвани системни ресурси, вървящи процеси и техните състояния, както и за активни сервизи в системата. И двете програми позволяват на потребителя да променя приоритета и affinity маската на програмите активни в момента, но позволяват за по-сложни операции.



Фиг. 8 - Windows Task Manager под Windows 10.

WTM и HTOP предлагат следните предимства:

- **Вградени част от операционната система** - Windows Task Manager е част от операционната система от повече от 25 години и HTOP е включен в повечето дистрибуции на Linux.
- **Гъвкави и потребни за всеки клас потребител.**



Фиг. 9 - HTOP под Xubuntu 20.04.

WTM и HTOP споделят следния недостатък:

- **Не са предназначени за автоматизация** - В основата си операциите, които позволяват тези две софтуерни решения са предназначени за единични случаи.

Bash и Powershell скриптове

Всички модерни операционни системи имат командни интерфейси за операция без графична среда, автоматизация или просто заради legacy поддръжка, именно възползвайки се от тези системи, създаването на скриптове за автоматизация и решаване на подобни проблеми е възможно, но за поддръжката и употребата на тези скриптове се изисква потребителя да бъде наясно как работят подлежащите им системи.

```
1   procs = "firefox", "notepad", "game";
2   ForEach($procname in procs)
3   {
4       ForEach($PROCESS in GET-PROCESS processname)
5       {
6           $PROCESS.ProcessorAffinity=255
7       }
8   }
```

Фиг. 10-1 - PowerShell скрипт за прилагане на affinity маска на избрани процеси по име.

Скриптове предлагат следните предимства:

- **Няма нужда от външни библиотеки** - Използват вградени функции за дадената операционна система.
- **Компактни** - В повечето случаи е достатъчно просто да се копират от една машина в друга за да работят.
- **Отворен код** - Скриптовете се интерпретират, така че кода е видим по всяко време.

Скриптове страдат от следните недостатъци:

- **Трудно за поддържане и промяна**
- **Изисква по високо ниво на компетентност** - В повечето случаи се изисква ниво на компетентност на начинаещ програмист или системен администратор, за да може да се използват ефективно.

```
#!/bin/sh
exec taskset 1 /bin/gzip "$@"
```

Фиг. 10-2 - Bash скрипт за прилагане на affinity маска на един процес чрез пътя в системата на този процес.

Заключение на анализа

Очевидно е, че има място на пазара за междинно решение, компактно софтуерно решение предназначено за всеки тип потребител, предлагащ лесна употреба и отворен код, за да позволи бързи модификации.

Ако трябва да категоризирами горните решения по три критерии:

- Автоматизация
- Достъпност
- Отворен Код, лесно за модификация.

Всеки един от тях покриват не повече от две от тези критерии.

Проектиране на ново решение

Очевидно е, че ако има каквато и да е нужда от ново решение то ще бъде за нещо, което покрива базите, пропуснати от вече съществуващите решения. От анализа се вижда, че това ново решение ще трябва да има следните качества:

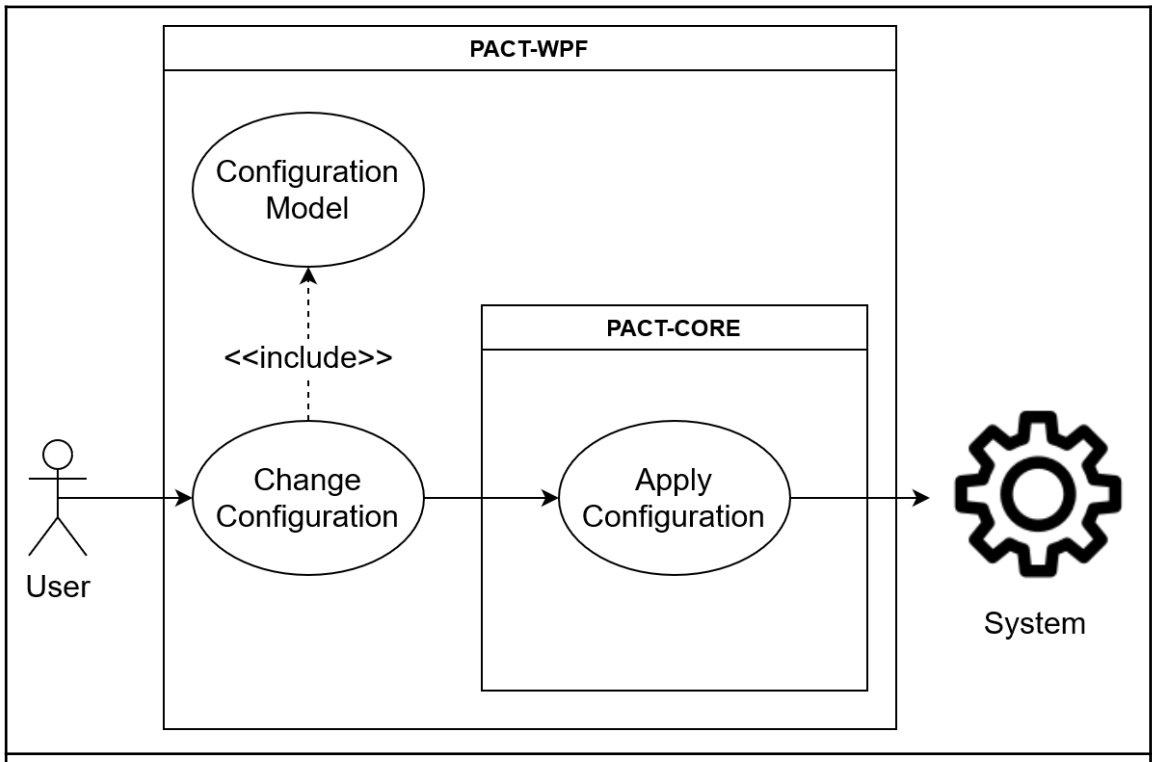
- Да автоматизира управлението на Affinity маските и приоритети.
- Да има фокусиран и лесно за употреба потребителски интерфейс.
- Да е гъвкав за употреба от други разработчици на софтуер и сорс кода да е достъпен за всички.
- Да поддържа, поне в основата си, възможно най-много модерни операционни системи и процесорни платформи.
- Да е портативно, конфигурациите да се преизползваеми от между операционни системи и компютри (с подобни процесори по ядра/нишки).

Основни Use Case-ове

За достигане на горе-изброените цели, нужно е да се покрият няколко основни use case-ове. Тези операции биват стартирани от потребителски интерфейс, който разчита на библиотеката за управления на процеси в операционната система.

Конфигуриране на Процес или група.

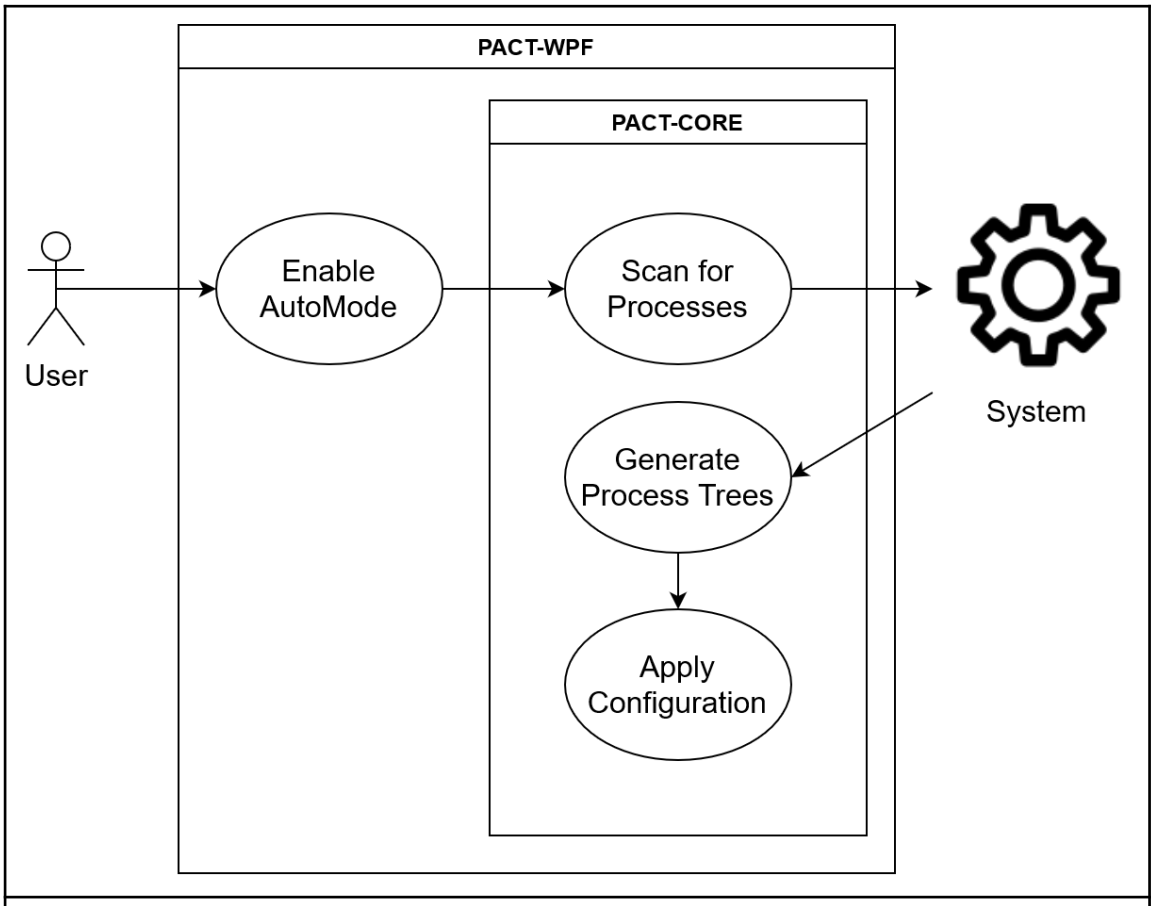
Потребителя трябва да може да прилага конфигурация за индивидуални конфигурации за процеси или общи за групи. Този процес не трябва да се различава, за да може да бъде интуитивен.



Фиг. 11-1 - Use Case диаграма за промяна на конфигурация на даден процес или група.

Автоматичен режим.

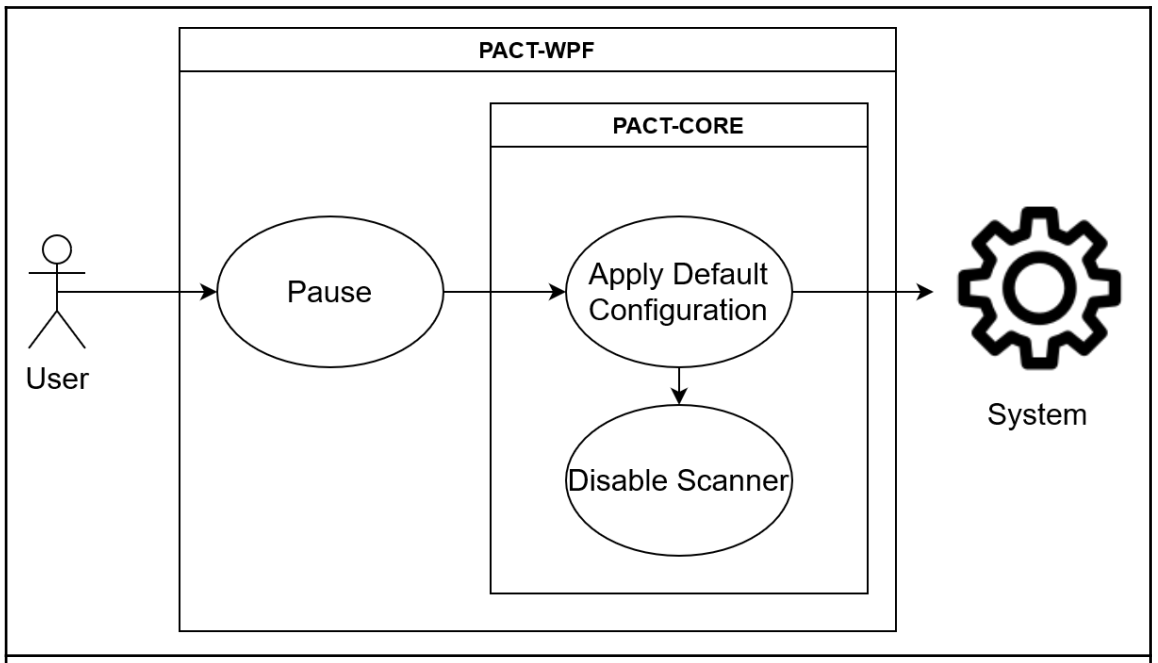
Имплементацията на системи за автоматизация, по-лесен за употреба чрез интелигентно сканиране на процеси и създавана не процесни дърва. Употребата на тези дърва за прилагане на конфигурации се очаква да позволи на потребителя да не създава конфигурации за всеки различен процес.



Фиг. 11-2 - Use Case диаграма за включване и операция на автоматичния режим.

Възстановяване на нормалния режим на работа.

По време на операция, потребителя може да реши или да се нуждае от това да спре процесите на сканиране и да възстанови нормалната операция на операционната система, следователно този случай също трябва да се покрие.



Фиг. 11-3 - Use Case диаграма за спиране на нормалните операции на програмата..

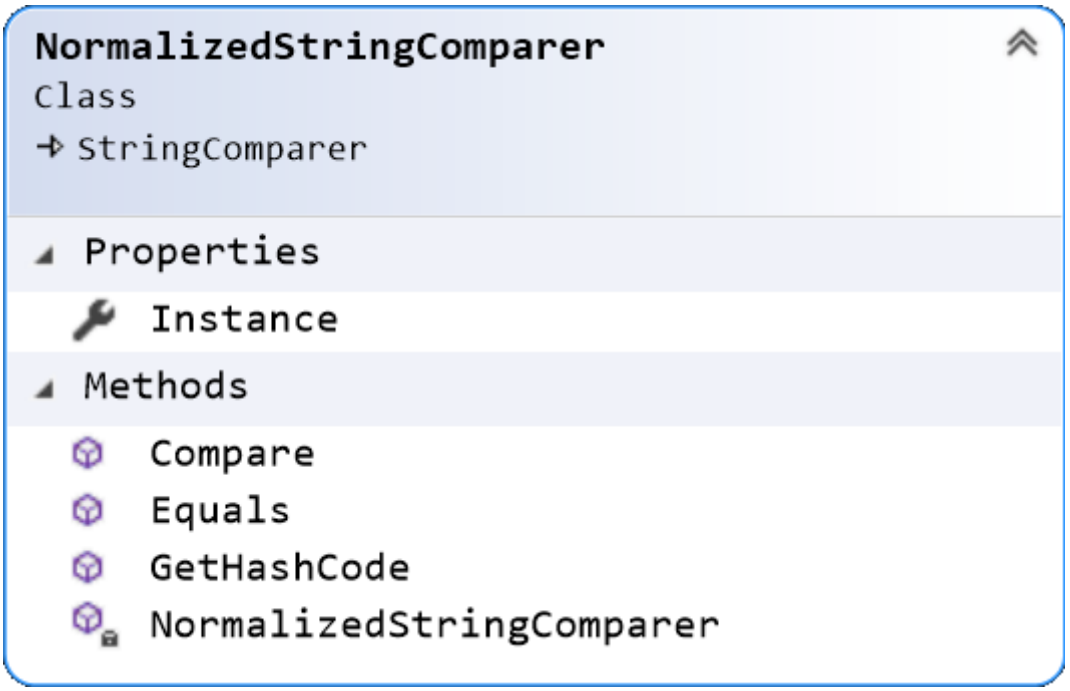
PACT-Core - Предназначение и Класове

PACT-Core е предвиденото решение, предназначено да притежава качествата изброени по-рано. То се състои от следните класове:

- **Помощни:**
    - CaseInsensitiveDictionary
    - CaseInsensitiveHashSet
    - NormalizedStringComparer
    - PACTInstance.
- **Моделни:**
    - PACTConfig
    - ProcessConfig
- **Действащи:**
    - ProcessOverwatch

NormalizedStringComparer

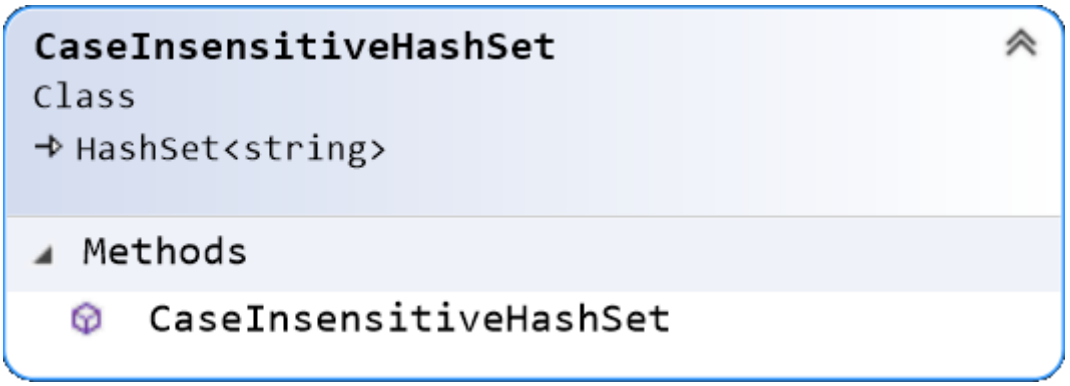
Клас, имплементиращ компаратор за символни низове за употреба от други линейни структури от данни. Причината да има нужда от такъв компаратор е да се избегнат ръчните проверки при употребата на речници и хеш таблици, тъй като операциите им са зависими основно от стойности, получени от хеширане на имената на процеси, които могат да имат различна капитализация.



Фиг. 12-1 - Клас диаграма за NormalizedStringComparer.

CaseInsensitiveHashSet

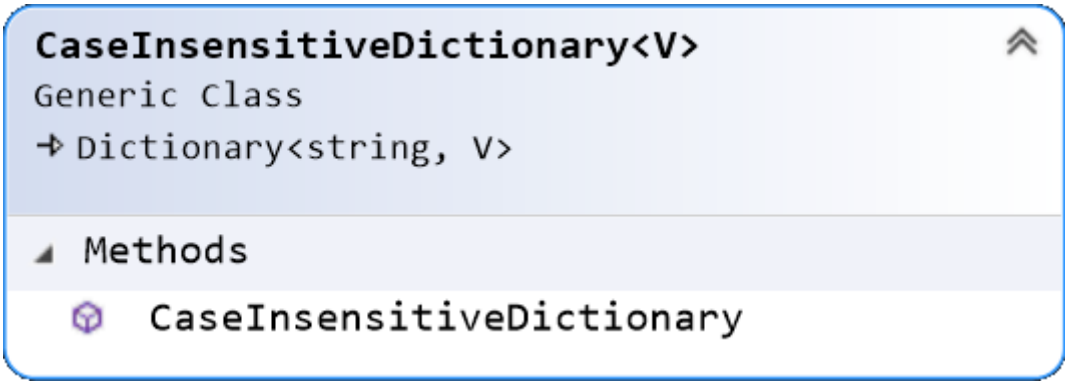
Хеш таблица, конфигурирана да игнорира капитализация на символни низове. Използва се за пазене на списък от процеси, които вече са сканирани.



Фиг. 12-2 - Клас диаграма за CaseInsensitiveHashSet.

CaseInsensitiveDictionary

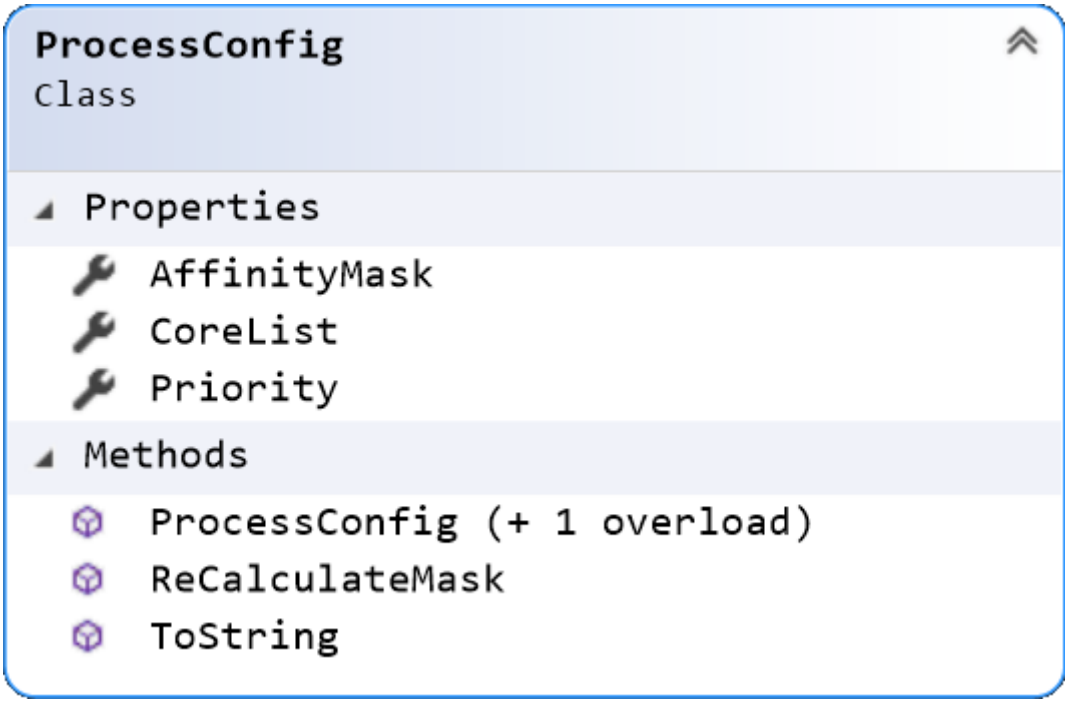
Речник, конфигуриран да игнорира капитализация на символни низове, когато се използват като ключ. Използва се на няколко места, за да пази процеси и конфигурации или конструкция на процесни дървета.



Фиг. 12-3 - Клас диаграма за CaseInsensitiveDictionary.

ProcessConfig

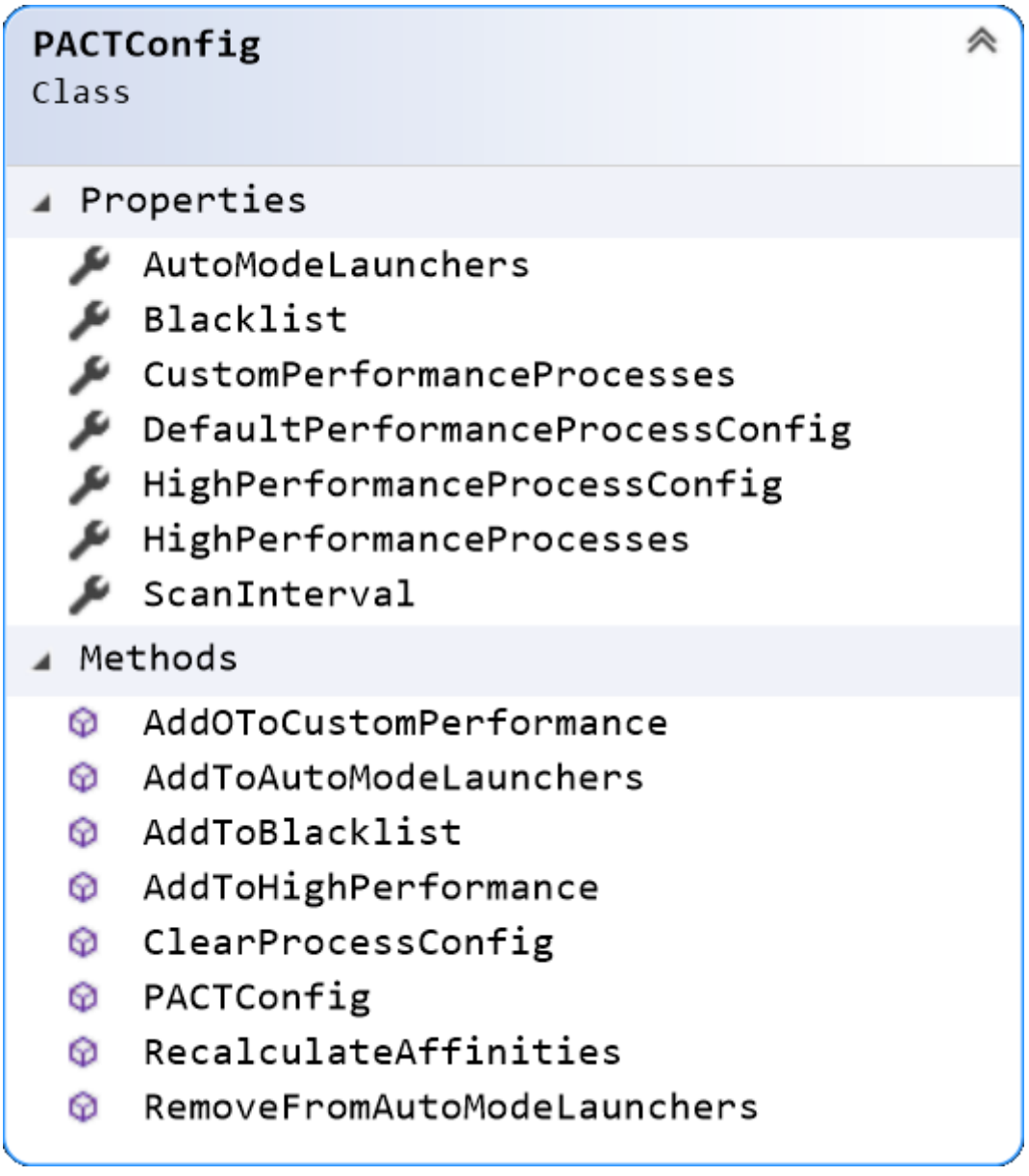
Конфигурация за индивидуален процес. Включва информация за къде и как трябва да работи един процес в системата (ядра, приоритет, и тн.). Може да се приложи на един или цяла група от процеси. Конфигурацията, на която разчита ProcessOverwatch, наречен PACTConfig, разчита основно на този клас.



Фиг. 12-4 - Клас диаграма за ProcessConfig.

PACTConfig

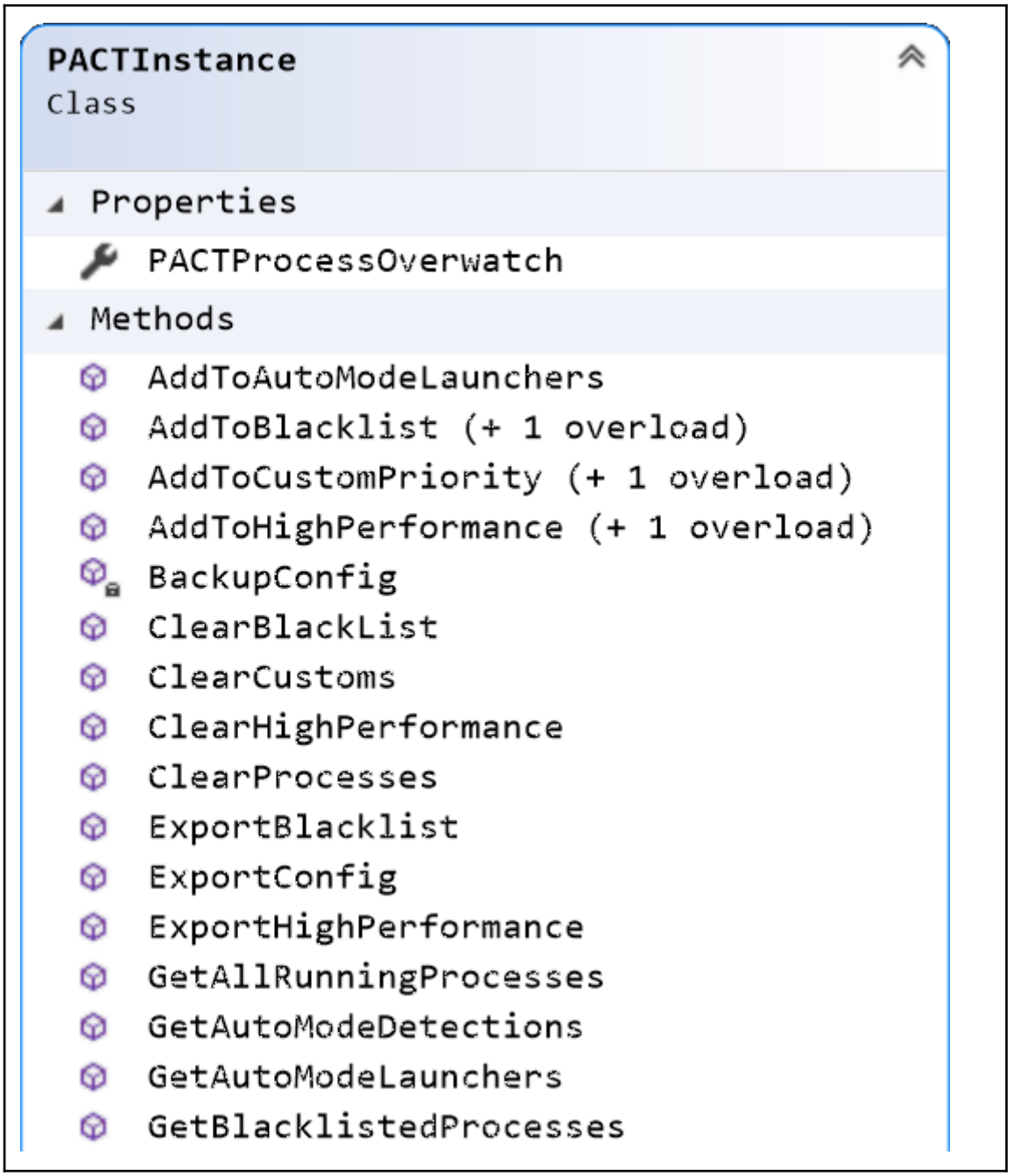
Моделен клас за цялостната конфигурация на PACT. Вътре са включени всички конфигурации, имена на процеси и други данни. Предназначен е да се записва и чете от JSON файл, но това е опционално.

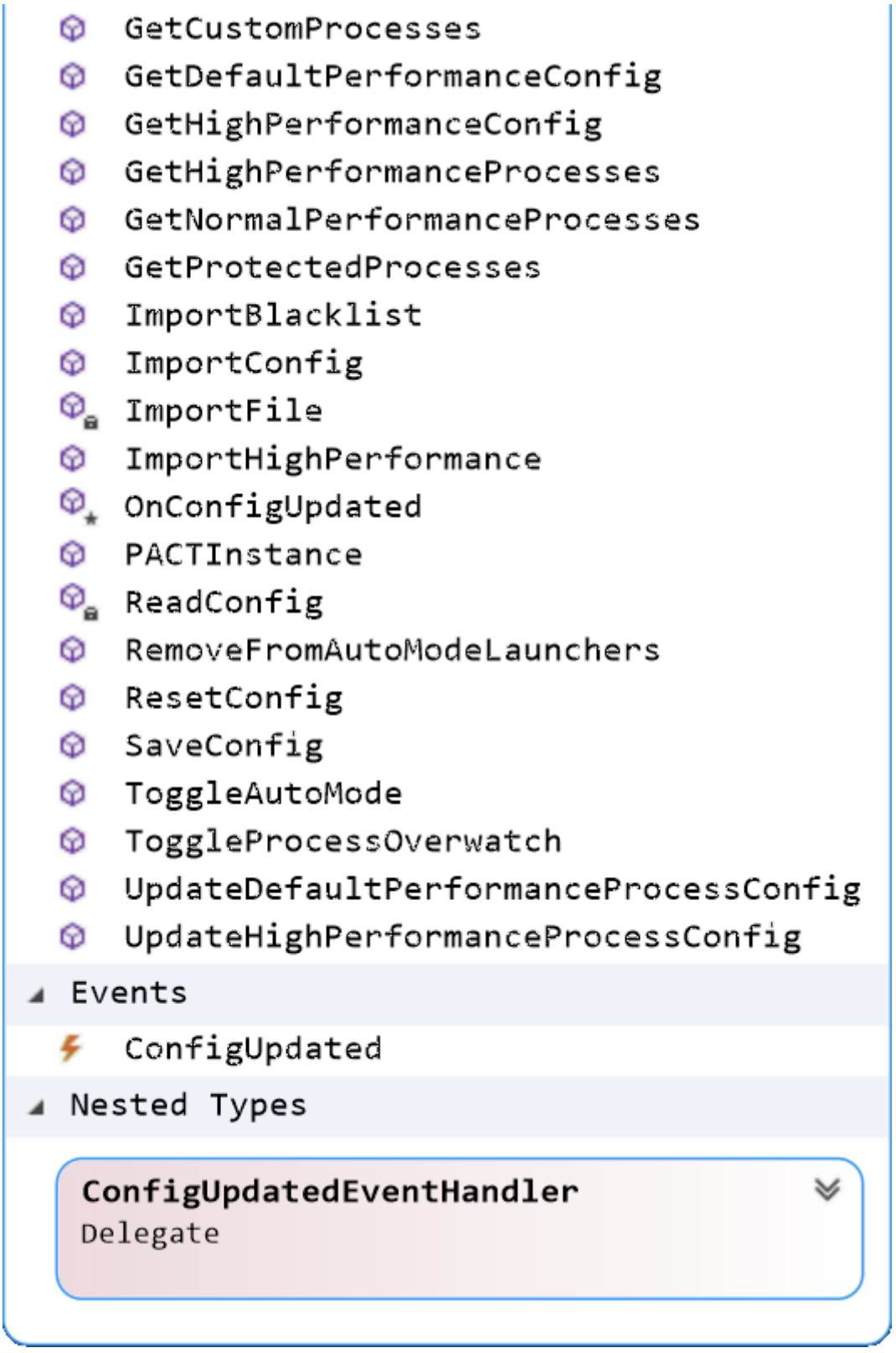


Фиг. 12-5 - Клас диаграма за PACTConfig.

PACTInstance.cs

Клас, предназначен да съдържа всички очаквани операции за един интерфейс, като методи, готов за употреба. Основно не е същински за операциите на PACT, но позволява създаването на по-компактни интерфейси като мести логическото управление на ProcessOverwatch в един клас. Този клас се използва от под-проекта PACT-WPF и е предназначен да се използва от PACT-Universal вбъдеще.

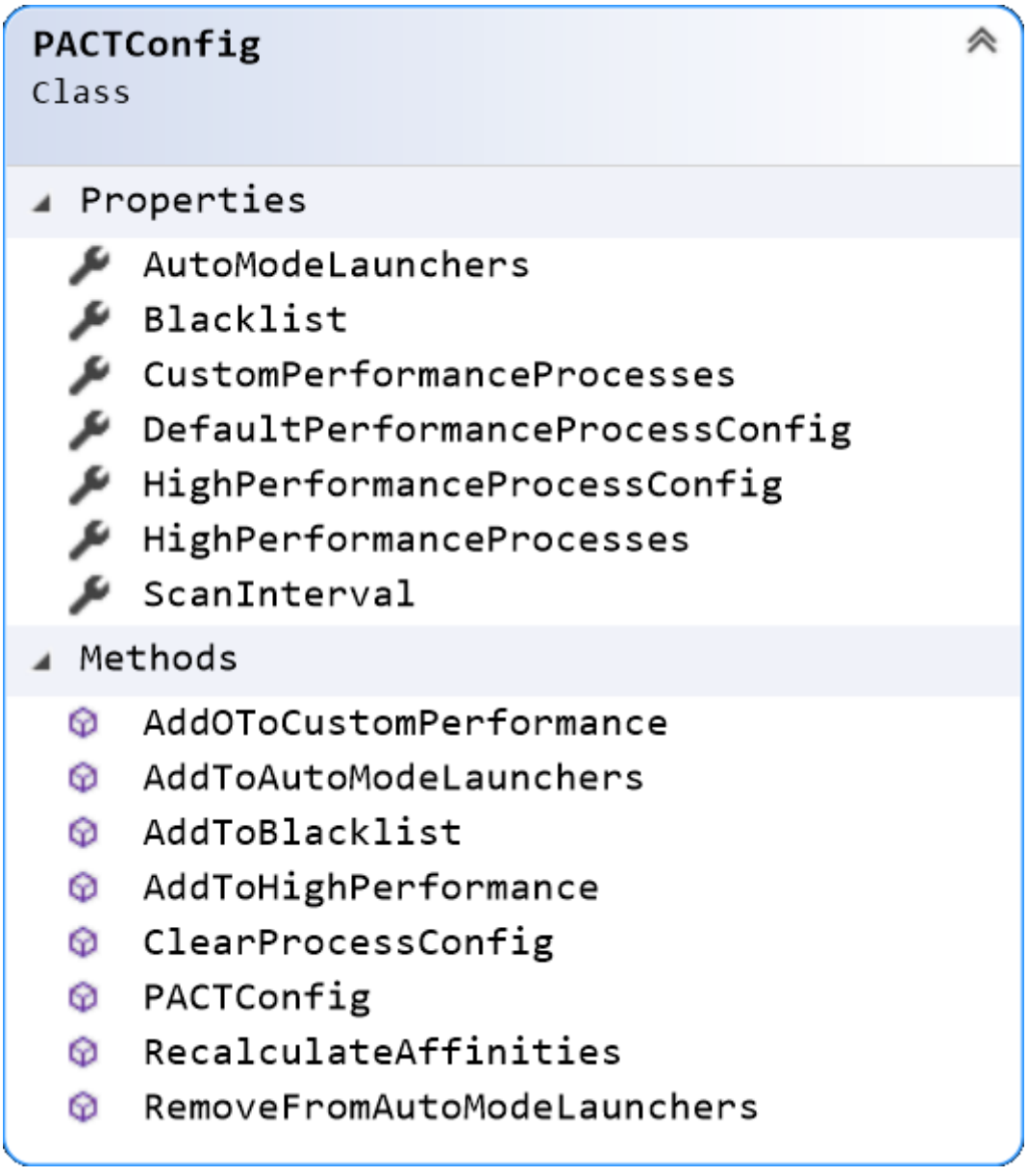




Фиг. 12-6 - Клас диаграма за PACTInstance.

ProcessOverwatch

Действащият клас в проекта, длъжен за сканиране на процеси, конструкция на дървета на процеси, разпределяне на всички процеси из нишките, според конфигурациите им.



Фиг. 12-7 - Клас диаграма за ProcessOverwatch.

P.A.C.T. като решение

PACT е акроним за "Process Affinity Control Tool". Има за цел да предостави удобен начин за разпределяне на процеси между ядрата/нишките на една компютърна система, било то истински хардуер или във виртуална среда. Предназначен е да бъде лесен за употреба от потребители, удобен за адаптация от програмисти и компактен, за да бъде лесен за преизползване.

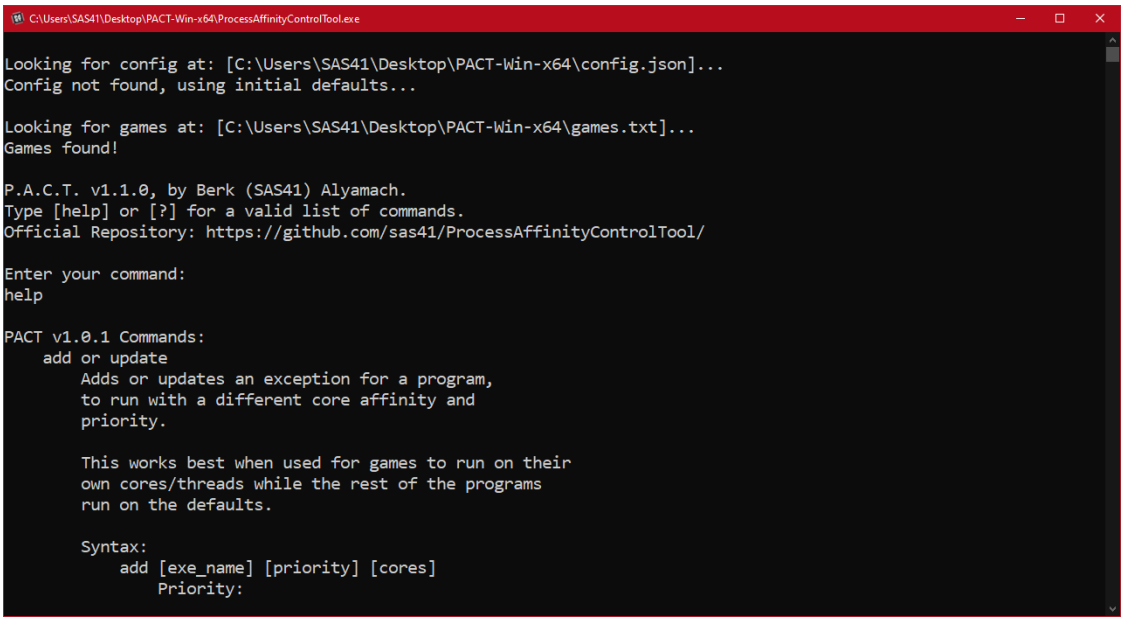
PACT V1.0

Първата версия на PACT е просто доказателство за идеята, че е възможно да се напише безплатна алтернатива с отворен код на платени алтернативи с .Net/C#.

Версия 1.0 се счита за Legacy, но все още е достъпна, тъй като конзолната версия на V2.0 все още е в разработка.

V1.0 има няколко големи недостатъка, тъй като е написан в рамките на един ден. Цялостният проект е монолетичен и не е разделен на библиотека и интерфейс, но работи под Windows, Linux, MacOS с x86, AMD64 и ARM базирани системи. Конфигурационния файл може да бъде преизползван, както и списъка за “игри”. Той може да съдържа всякакви процеси вътре, не само игри, така че може да се адаптира по нуждите на потребителя.

Тази версия поддържа само интерактивен режим на употреба с идеята, че ще работи на един от терминалите в задния план.



Фиг. 13 - Интерактивния интерфейс на PACT V1.1.0.

PACT V2.0

PACT V2.0 е почти цялостно преструктуриране и пренаписване не оригиналния проект. Проекта в момента е разделен на три основни части:

- **PACT-Core** е библиотеката която извършва основните системни действие за промяна на affinity маски и приоритети, сканиране за нови процеси и конструкция на процесни дърва.
- **PACT-WPF** е графичен интерфейс за Windows операционни системи, като предоставя потребителски интерфейс за управление и адмистрация на конфигурации.
- **PACT-Universal** е терминален/конзолен интерфейс за всички операционни системи и архитектури поддържани от .Net Core. Тази част на проекта е все още в разработка и е предназначена да бъде истинския наследник на PACT V1.0.

Функционалности на PACT

До момента групата от проекти PACT предлага са следните функционалности:

- Автоматично прилагане на affinity маска на даден процес, според конфигурацията и групата му.
- Автоматично повишаване или понижаване на приоритета на даден процес, според конфигурацията и групата му.
- Масово прилагане на конфигурации за процеси с три основни категории.
- Индивидуални конфигурации за всеки даден процес по избор.
- Автоматично откриване на процесни дърва и прилагане на конфигурации според стартиращия процес.
- Импортиране и експортиране на отделни части от конфигурацията и/или цялата конфигурация.

Използвани технологии и библиотеки

Проектът е основно базиран на .Net Core технологиите и е написан на програмния език C# от Microsoft. Следните части от .Net Core стека са използвани в проекта:

- **PACT-Core** (C#):
    - .Net Core Standard Libraries
    - Microsoft.System.Diagnostics Library
- **PACT-WPF**(C#):
    - .Net Core Standard Libraries
    - Windows Presentation Foundation Standard Libraries
    - Windows Forms Standard Libraries
- **PACT-Universal** (C#):
    - TBD

Технически детайли на PACT-Core

PACT-Core има две основни действащи системи, чрез който управлява разпределението на процеси в системата:

- **Активно сканиране.**
- **Автоматичен режим.**

Активно Сканиране:

Активното сканиране за живи процеси в системата. Прави се списък на всички процеси, работещи в системата, ако те се намират в дадена категория (Висока Производителност, Индивидуална Конфигурация, Черен списък.), следва дадено действие:

1. Ако процесът е в черния списък, той се игнорира.
2. Ако процесът е в категория "Висока Производителност", се прилага обща конфигурация за ВП.
3. Ако е с индивидуална конфигурация, се прилагат съответните настройки от тази конфигурация.
4. Ако нито един от горните правила не важи, се прилага стандартната конфигурация.

Автоматичен режим:

Повечето платформи за видео игри и софтуерна дистрибуция за немобилни операционни системи стартират игри и програми като дъщерни процеси, следователно те имат особени правила. Примерна ситуация може да бъде следната (стартиране на игра чрез Steam):

1. Steam стартира "Launcher" за играта.
2. "Launcher" стартира "Anti-Cheat" софтуера.
3. "Anti-Cheat" софтуера стартира процеса на играта.

Подобни на това потоци се виждат и при софтуерни приложения с DRM или конфигурационни интерфейси, преди самата апликация да се пусни.

Проблемът тук е че много често тези Anti-Cheat и DRM решения, използвани от тези апликации, не позволяват да се конфигурират действащите процеси под тях и тези процеси много често наследяват конфигурацията си от предишния процес във веригата/дървото от процеси.

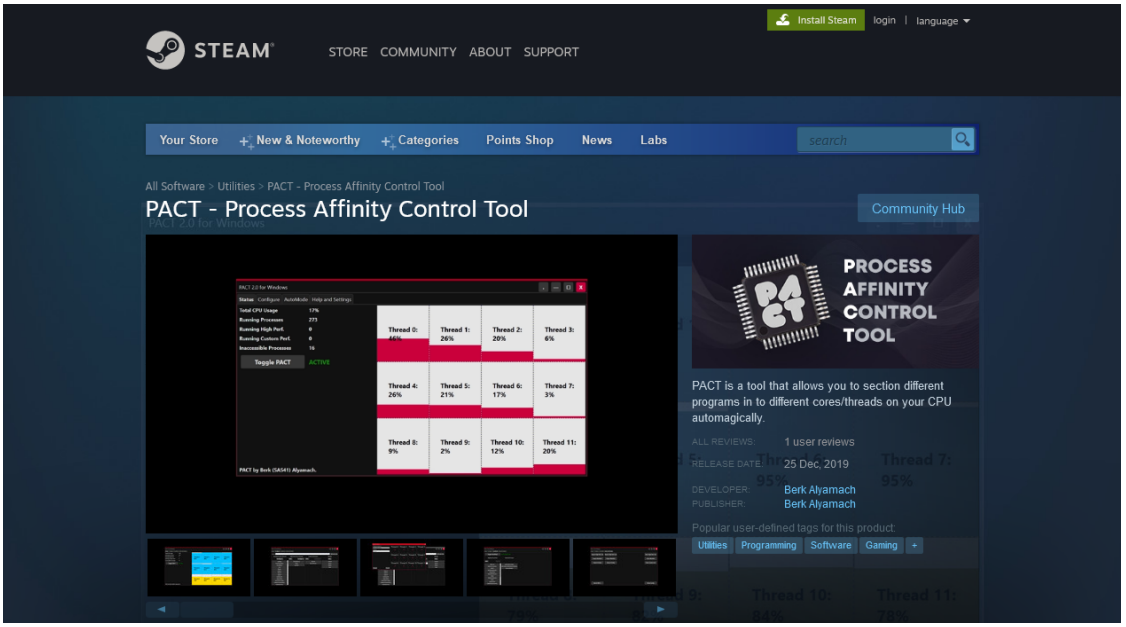
Автоматичния режим позволява на потребителя да избере процеси, които често извикват други процеси и прилага конфигурация за висока производителност на дъщерните процеси, за да избегне ситуации, където основния процес не може да бъде конфигуриран, но е наследил конфигурацията от родителския процес, следователно може да има ниска производителност.

В списъка се вкарват потенциални родителски процеси. Това премахва нуждата да се конфигурират повечето процеси индивидуално. Потребителят, независимо какъв софтуер използва от дадена платформа, може да бъде сигурен, че процесът ще върви с висока производителност напълно автоматично.

PACT-WPF

PACT-WPF е предназначен да бъде графичен потребителски интерфейс за Windows операционни системи. Използван е Windows Presentation Foundation като основа на графичния интерфейс и се използва PACT-Core библиотеката за основни операции.

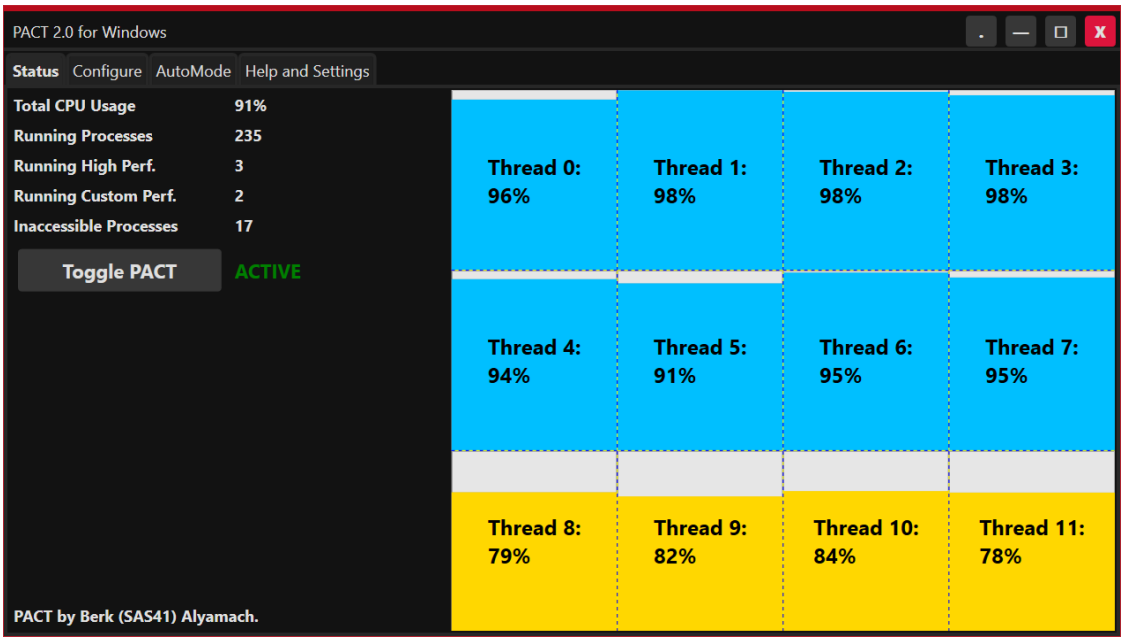
PACT-WPF също се предлага в компилиран вариант в платформата за онлайн софтуерна дистрибуция Steam от Valve.



Фиг. 14 - PACT в Steam (store.steampowered.com/app/1517370)

Графичния интерфейс на PACT-WPF

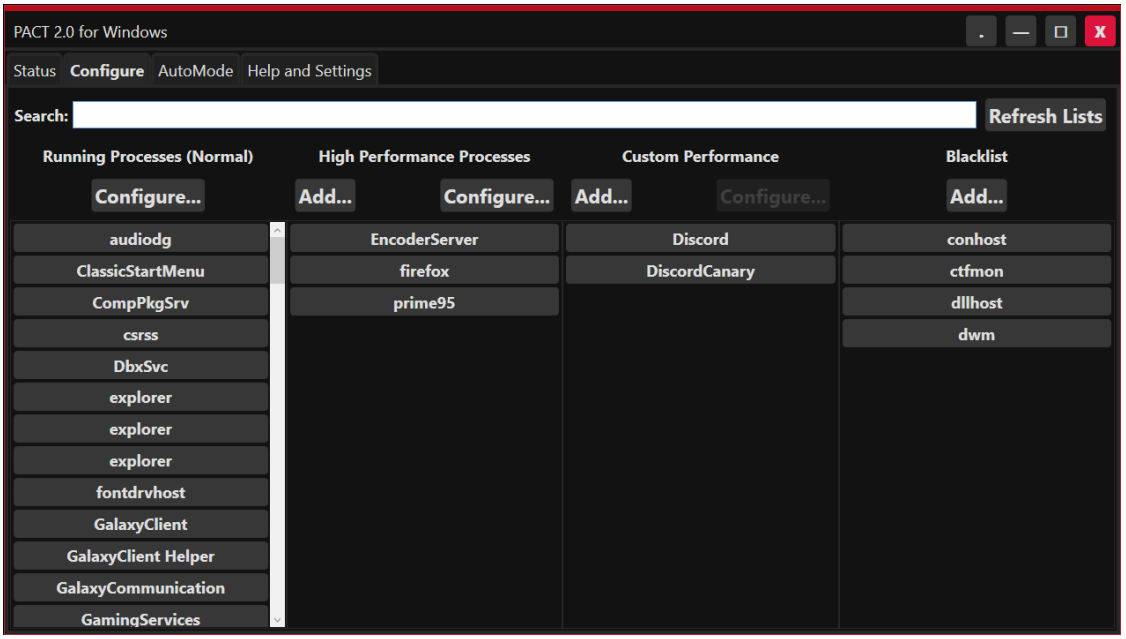
Графичния интерфейс на PACT-WPF позволява достъп до всички основни функционалности на PACT-Core библиотеката, както и информация относно текущото състояние на програмата и повърхностен изглед на процесора на системата.



Фиг. 15-1 - Главната страница на графичния интерфейс на PACT-WPF.

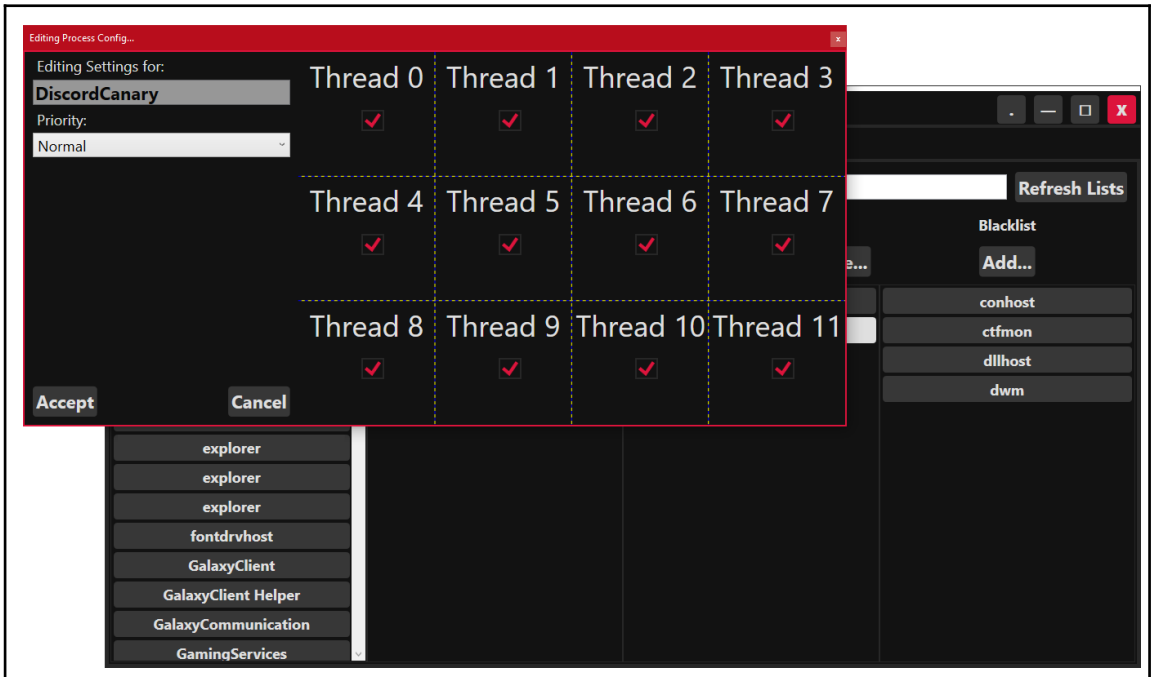
Главната страница е предназначена да информира потребителя относно текущото състояние на тяхната система. Бутоните, налични при всички екрани горе, позволяват на PACT да бъде минимизиран, максимизиран, затворен, или да бъде скрит при иконите в долния десен ъгъл. Информацията включен тук се състои от следните елементи:

- Общата употреба на процесора.
- Общия брой процеси активни в момента.
- Активни процеси с приложен профил “High Performance”.
- Активни процеси с приложен индивидуален профил.
- Активни процеси, недостъпни от PACT, системни и подобни



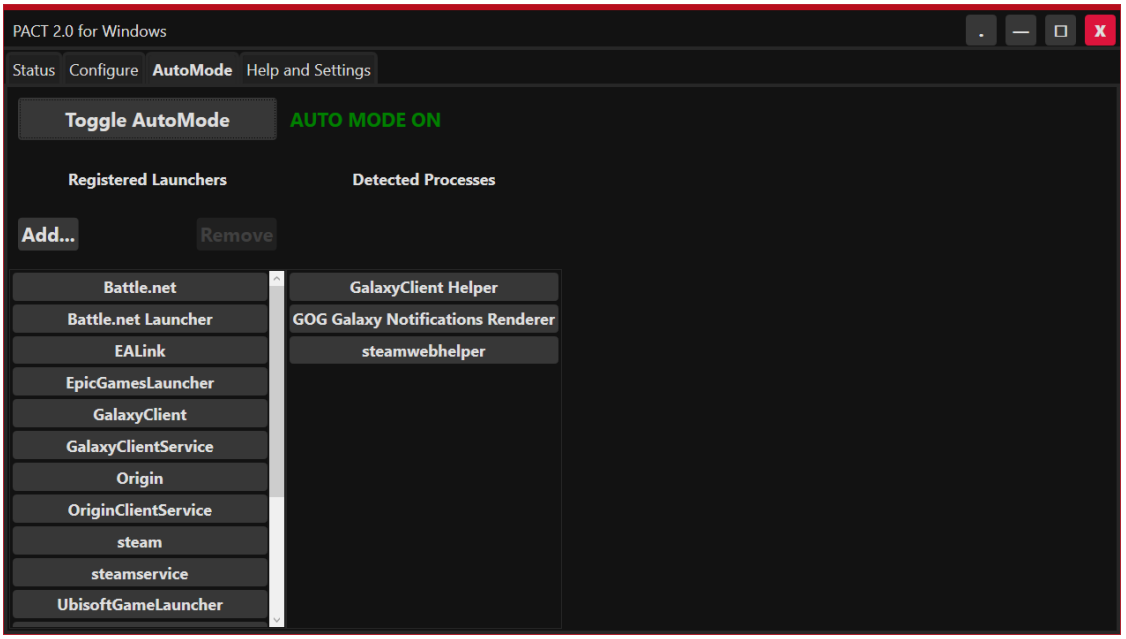
Фиг. 15-2 - Конфигурационната старница на PACT-WPF за Windows.

Основната конфигурационна страница позволява на потребителя да мести процеси от една група в друга чрез Drag & Drop. Тук потребителя също може да промени индивидуалните конфигурации за всеки процес или да промени конфигурациите за двете обобщителни групи. Също така, процесите могат да бъдат преместени в черния списък. PACT ще игнорира процесите в черния списък. Търсачката позволява филтриране на процесите във всички категории.



Фиг. 15-3 - Конфигурационния прозорец за групи и индивидуални процеси.

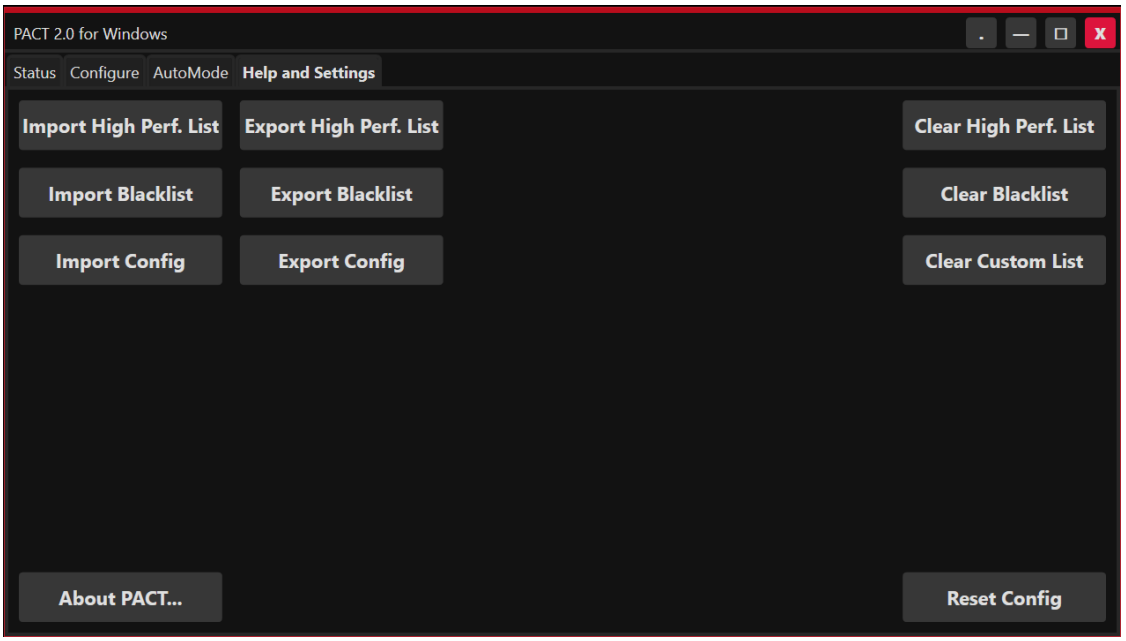
Прозореца за профилна или индивидуална конфигурация позволява да се изберат индивидуални ядра/нишки, на които процеса или процесите от дадена категория могат да бъдат изпълнени. Също така позволява да се избере приоритета за тази категория или процес.



Фиг. 15-4 - Конфигурационната страница за автоматичния режим.

Автоматичния режим има за цел да конструира процесно дърво и да открие, ако даден процес е стартиран от друг процес. По този начин то може да прилага дадена конфигурация автоматично, без никаква намеса от потребителя, единствено е нужно да знае за кои майчини процеси да търси.

Тук се намира списъкът на майчините процеси, към които могат да се добавят или премахват елементи, както и намерените дъщерни процеси. Бутонът най-отгоре позволява спиране на сканирането.



Фиг. 15-5 - Страницата за импортиране, експортиране и изчистване на разни конфигурации.

Последната страница съдържа бутон за информация и други бутони за експортиране и импортиране на конфигурации и списъци, както и бутони за изчистване на тези списъци и конфигурации.

Бутона за информация води до GitHub страницата на проекта.

Примерни употреби на PACT

PACT е предназначен основно за разпределение на програми по нишки за изпълнение. Нуждата от това е очевидна при следните ситуации:

Аудио обработка в реално време

Много често аудио обработката в реално време изисква минимални закъснения и, ако едно ядро трябва да сподели време за изпълнение с няколко процеси, не е сигурно дали драйверите ще получат време за изпълнение, когато е нужно.

3D в реално време

Най-често при видео игри, консистентното време между кадри е един от най-важните фактори за такъв вид софтуер. Това оставя само 16.66 милисекунди за всеки кадър, при 60 кадъра в секунда, но за играчи в турнири и ентусиасти, 144 или 240 кадъра са популярни опции, което означава, че всеки кадър има ~8, ~4 милисекунди. В тези ситуации всеки цикъл на процесора е от значение.

Видео обработка в реално време

Транскодиране на видео в реално време е изключително тежка операция и в много случаи, ако процесорната мощност е недостатъчна, изменения и забавяния във видео и аудио се срещат. Този проблем най-често се вижда при по-маломощни процесори. Също така при живо излъчване на видео игри, делегацията на нишките между процеса на "Rendering" и "Streaming" е много важен за гладки резултати.

Селективно бързодействие на натоварени системи

За някои потребители, системата може да е под постоянен стрес, който води до флегматична операция и ниско бързодействие, дори за критични операции като системна администрация, проблем, който се среща често в сървърни среди. Именно за да се предотвратят такива ситуации, е добра идея някои процеси да се маркират с по-висока важност и да може тази йерархия да се промени в движение.

Системна сигурност

В редки случаи, при стари архитектури, някои от критичните бъгове за сигурност разчитат на операцията на процесора да прескача между няколко процеса и да извлича информация от кеша на индивидуални ядра. По този начин, ако критични операции биват изолирани, теоретично този метод на експлоатация може да се избегне.

Примерни конфигурации за хипотетични ситуации

В разни примерни системи, PACT може да бъде конфигурирано по следните начини за максимална ефективност:

За живо излъчване на видео игри (8 ядра и 16 нишки)

- 6 Нишки за Видео игра.
- 4 Нишки за Транскодиране и живо излъчване.
- 2 Нишки (1 ядро) за аудио обработка в реално време.
- 4 Нишки за всички други процеси с висок приоритет на по-важни програми като браузър или комуникационен софтуер.

Това осигурява консистентно време между кадрите и гладко видео за излъчване, без отхвърлени кадри в транскодера.

За живо Транскодиране в медиен сървър (8 ядра и 16 нишки)

- 8 Нишки за транскодиране в реално време.
- 4 Нишки за живо излъчване.
- 4 Нипки за всички други процеси, включително системни.

Осигурява гладко видео, без загубени кадри, или забавено аудио/видео, без нужда от буфериране.

За Build сървър

- Remote Management софтуера с най-висок приоритет на отделно ядро.
- Контролния панел с най-висок приоритет на отделно на отделно ядро заедно с горното.
- Build Manager с висок приоритет.
- Останалите програми с нормален приоритет.

Осигурява контрол над сървъра, дори други процеси да изискват много мощност.

Тестване и резултати

За тестването на ефектите на PACT, е използван примера за “Видео обработка в реално време”. Целта на този тест е да се проследи какви подобрения предлага PACT срещу нормалния режим на една Windows операционна система.

Методология

Тестването се състои от четири фази:

1. Тестване без системно натоварване и без PACT.
2. Тестване без системно натоварване със PACT.
3. Тестване със системно натоварване но без PACT.
4. Тестване със системно натоварване и PACT.

Тестването се състои от енкдориане на живо, видео с висок bit-rate, използвайки процесора. За целта е използван **VLC Медия Плеър** като източник на видеото и **OBS Studio** за хващането и енкодирането на кадрите.

За системно натоварване е използван **Prime95,** който е добре познат в сферата на “Overclock”-ването като добър стрес тест за процесора, който също така използва много оперативна памет. Prime95 също така ползва AVX инструкции, които често се срещат при математически софтуер, и са известни с това, че натоварват процесора допълнително при някои обстоятелства (генерират повече топлина и карат процесора да се забави като термозащита, също познато като “Thermal Throttling”).

Използвайки комбинацията от тези приложни програми, може да се направи тест, демонстриращ екстремен случай където софтуер като PACT може да помогне с производителността на системата.

Оценката на всеки тест ще зависи от пропуснатите кадри в процеса на енкодирането. Тъй като енкодера разчита на процесора, за да обработва кадрите, ако процесора не може да отдели време за енкодирането, ще се пропуснат кадри. Критерия за оценяване ще бъде изпуснатия брой кадри през 30 секунди, като процент от общия брой кадри. Компютъра ще бъде рестартиран (Cold Boot) след всеки тест.

Системна Конфигурация

Процесор:	**AMD Ryzen R5 3600 (6C 12T)**
Честота:	**4200 MHz, all-core**
Памет:	**2x8GB DDR4 3600-CL 14,14,14,36**
Нос. на Данни:	**Samsung SSD 970 Evo Plus**
Дънна Платка:	**MSI B450 Tomahawk MAX**

OBS Настройки

OBS Version:	**26.1.1 - 64-bit Windows**
Video Encoder:	**x264**
Video Format:	**h264**
Resolution:	**2560x1440**
Mode:	**Progressive**
Frame Rate:	**60 FPS**
Bit-Rate:	**6000 Kbps**
Rate Control:	**Constant Bit Rate**
Keyframe Interval:	**Auto**
CPU Usage Preset:	**veryfast**
Profile:	**(None)**
Tune:	**(None)**

PACT Настройки

PACT е конфигуриран да раздели процесорните нишки на две групи. Първата група са първите шест нишки (от 0 до 5), където работят VLC Медия Плеър и OBS Studio, а втората половина с нормален приоритет е резервиран за Prime95. Останалите процеси с под-нормален приоритет са също при втората половина.

Резултати

Тест 1 - Тестване без системно натоварване и без PACT.

Пропуснати кадри: **0.0%**

Тест 2 - Тестване без системно натоварване със PACT.

Пропуснати кадри: **0.0%**

Тест 3 - Тестване със системно натоварване но без PACT.

Пропуснати кадри: **71.2%**

Тест 4 - Тестване със системно натоварване и PACT.

Пропуснати кадри: **44.5%**

Интерпретация на резултатите

От тестването се вижда че PACT, при тези крайни обстоятелства, може да има положителен ефект, но все пак, не е способен напълно да отстрани проблема. Нуждата от страна на Prime95 към оперативна памет, а не само процесорни цикли, ограничава колко може да се постигне, само чрез разпределяне на процесите. PACT не обещава положителни резултати при всички употреби и ситуации. В крайна сметка, всички системи са твърдо ограничени от производителността на техните процесори.

Заключение

PACT вече е напълно функционален и постигна всички поставени цели. Над 30 потребители го намериха за достатъчно потребен, за да заплатят за него, като до момента той не разполага с негативни отзиви. От анекдотични мнения на потребители и чрез емпирично тестване се вижда, че крайния резултат е ефективен в автоматизацията на управление на процеси.

Проектът успя да постигне зададените цели.

Литература

1. Въведение в Програмиране със C#
   Колектива на Телерик (2011)
   ISBN: 978-954-400-527-6

2. Fundamentals of Computer Programming with C#
   Светлин Наков и колектив (2013)
   ISBN: 978-954-400-773-7

3. Въведение в .Net
   Денис Колисинченко (2016)
   ISBN: 978-954-8898-86-7

4. Основи на програмирането със C#
   Светлин Наков и колектив (2017)
   ISBN: 978-619-00-0635-0

5. https://docs.microsoft.com/en-us/dotnet/
   Official .Net documentation by Microsoft.

6. https://docs.microsoft.com/en-us/dotnet/csharp/
   Official C# documentation by Microsoft.

7. https://docs.microsoft.com/en-us/visualstudio/
   Official Visual Studio documentation by Microsoft.

8. https://en.wikipedia.org/wiki/Multiprocessing
   https://en.wikipedia.org/wiki/Multiprocessor_system_architecture
   https://en.wikipedia.org/wiki/Multithreading_(computer_architecture)

9. The Quiz on CPU 0: Playing Scheduler Wars with AMD's Threadripper 2990WX
Ian Cutress (2018)
https://www.anandtech.com/show/13446/the-quiz-on-cpu-0-playing-scheduler-wars-with-amds-threadripper-2990wx

10. Threadripper 2990WX Performance Regressions? Not so fast…
Level1Techs, Wendell (2018)
https://www.youtube.com/watch?v=WSSAFqzbKgg

11. 2990WX Threadripper Performance Regression FIXED on Windows*
Level1Techs, Wendell (2019)
https://www.youtube.com/watch?v=M2LOMTpCtLA

12. https://en.wikipedia.org/wiki/Side-channel_attack

13. PortSmash Side-Channel Vulnerability CVE-2018-5407 information
Werner Fischer (2018)
https://www.thomas-krenn.com/en/wiki/PortSmash_Side-Channel_Vulnerability_CVE-2018-5407_information

14. New Intel CPU Flaw Exploits Hyper-Threading to Steal Encrypted Data
Swati Khandelwal (2018)
https://thehackernews.com/2018/11/portsmash-intel-vulnerability.html