LAZARUS.CONSORTIUM

Lazarus' "Laplace" La+ Operating System

LA+ OS ('LAPLACE' for short) is a Construct intelligence based Operating System. Specifically, LAPLACE excels with networked intelligence platforms - a broad classification describing everything from workstations, power-armor, advanced robotics, ships and even star-bases.

While the software was originally distributed only privately as an operating environment for the Lazarus Consortium, the platform has matured and become sufficiently versatile for public use and entered public use following the UOC Open Source Licence in mid-YE36.

About

WIP: This article is approved for usage in the RP.

LAPLACE is derived from the open-source Regionless Inter-Configuration Environment or 'RICE' kernel which was written by Kage Yaichiro. Most of its components were built by Aiesu Kalopsia, drawing inspiration from Polysentience style network topologies and programming practices as well as academic examinations of the work of Hanako Ketsuri.

In terms of requirements, LA+ OS self-compiles on launch with no special knowledge and can run on any tier-2 computer but is designed to scale properly all the way to a Tier-21 performance platform and potentially beyond.

LAPLACE explicitly refers to the large collection of software packages and repositories built ontop of RICE. The OS follows a rabbit-motif in its design and many of its utility-names including its core cognitive programming language, LAPINE (LAPlace Integrated Neural Engine), a dialect of SCRIBOL. As an easter-egg, it also includes an unabridged copy of ' THE ART OF NEVER AGAIN' as a mark of respect.

LAPLACE from a user-standpoint

The majority of apps running in LAPLACE don't have a visual 'interface' of their own: Rather the interface is built up based on the user's requirements and involved datasets, meaning the same program can work just as well say in a cockpit as it can on a workstation, augmented reality, non-visual computer/brain neural interpreter or large volumetric display. Writing its own drivers and updating its own display libraries based on usage, the software understands user habits and builds upon them, while working with off the shelf, specialist or military technology.

Common Usage Case

Commonly in workstation mode, the software draws a conventional interface for ease of use. In Augmented Reality Mode, the interface is similar to that of a power-armour HUD. The most common usage case is as a combined workstation/Augmented Reality/Neural system, putting different information in different places.

Volatile Cognition

Importantly, the system understands and inter-operates basic cognitive concepts like language, quanta, space, mass and both objective and subjective comprehension models not only within itself but also from the user if a comprehensive neural interface is available. What this effectively means is the system understands what its actually being told to do and isn't just crunching random numbers on a processor – with varying levels of abstraction based on the available CPU power the same way a person would. This information is stored in high-level volatile memory under obsfurication, meaning it cannot be extracted from the machine and is instead generated per usage case.

Low-Level Access

In special cases, a 'root' level access is available for low-level command. The system both deals with information as files but also as indexed networked relative-depth-model cognitive quanta, similar to what many databases try to emulate with some information being properties or sub-properties of others.

Data-model

All information is treated not only with a per-user-permission/per-user-ownership model of information but also through a third system called keyed encoding – essentially meaning a file or non-file can be scrambled and re-ordered for feedback with different systems. Combined with high-level neural systems, this allows a user's own non-modified organic brain to become their hard-drive and their own willing consent without duress to become their password.

Actual decoding of this information by the user without any computer whatsoever is possible through MOTIVIA based training programmes, thought the information will not be interlinked, weighed or valued neurally (all part of common non-linear recall) until the information has been used.

Networked Operational Model

LAPLACE makes a key distinction between client-actions and server-actions. Client-actions happen on the side of the end user who is asking questions and executing things locally. The definition of server is blurred somewhat in that much of the heavy-lifting of a given machine (provided a quantum modem is

available) can happen remotely, making the end-user's machine essentially just a client.

Common practice is for users to build their own servers in secure locations and apply this practice, with a low-end device acting as a dumb-terminal for a much larger machine which does the real work.

Lazarus Processing Network

The LPN is literally communistic processing power: Ordinarily, a computer isn't particularly stressed most of the time, only working flat out at its peak maximum somewhere between 10% to 30% of its active life-span. The other 70-90% of available processing capacity not used is ordinarily wasted. The LPN lets a user give what would ordinarily be wasted and in return during that 10-30% peak, be granted extra push from the network.

This works using a very simple mode: The more a user donates to the network, the more the user can take from the network, building up a rapport of trust with the network over time, relative to the current average CPU time available to each client. At peak times, this trust is what discerns the available CPU power to be donated.

From an end-user standpoint, the result is that low-end devices, provided they are fitted with a quantum modem and LPN access can now perform actions associated with higher performance computers: A performance increase without actually modifying the device itself in any way.

In a nutshell, the LPN is cloud time-sharing.

Safety

For security reasons, all instructions are obsfuricated in such a way that they can not be recognised by the remote computers as meaningful information, only low-level instructions executed redundantly. In the end, the redundancies are compared to make sure the answer exists and has not been tampered with and only the end client understands what the data means or what the process involved to produce it actually was.

Persistence, addition & subtraction

Users can tell a program to become a daemon: A thing which happens in the background with low-CPU demand which can be accessed by the user using their Keyed Encoding. Even if the user's device is destroyed, a daemon keeps working provided the user has membership, provided the user or someone on their behalf anonymously continues to donate CPU runtime.

This same principle also applies to data. Importantly, if information is made public, the keyed encoding (which requires the user not be harmed or under duress) is the only thing which can be used to take the information back from the network. For this reason, information can be assigned multiple keyed encoding sources as a backup redundancy through a wide variety of security schemes (turn-key multi-user permission being popular). Time of Death/Cognitive Confirmation of Demise is also popular as a trigger

for not only the removal of data but for the release of data which was otherwise obscfuricated, offering the potential to act as an insurance policy for an end-user.

Programming

LAPINE is essentially a re-imagining of Phoenix' Arms SCRIBOL, using modern concepts from a variety of other programming techniques and neural syntax. It can effectively be described as “SCRIBOL without the scribbling”.

The name stands for LAPlace Integrated Neural Engine – meaning essentially that the code can be thought of as a sculpture which is shaped by the variables it deals with, within a given context. The compiled application has through construct-style programming and supplies LAPINE with an understanding of what its output should look like and what constitutes a good or a bad response. As time goes on, the program is educated case by case. Unlike people, the software can deal with very low-level abstraction and isn't prone to generalisation flaws or cognitive biases.

From an engineering standpoint, LAPINE dispenses with a lot of common elements of other programming languages which are platform specific or unsafe (such as specialised pointer and memory recall behaviour which is known to be exploitable) and instead operates with no assumption of memory/processing independence, assigning its own variables to processes on an intelligent case-by-case basis at the point of compile with the option for manual override in the source-code.

Essentially working with LAPINE is closer to the experience of briefing a programmer which is constantly evolving and getting better, rather than the experience of actually puttering around with code and hoping for a response. To this end, LAPINE's purpose is to democratise programming: Provided the user is capable of logical sequential thinking and can explain what is happening and what they want, LAPINE will work to provide a solution. More advanced users can then suggest directions to be taken to achieve a solution or even pick apart solutions during debugging and suggest optimisations or improvements.

Visualization

Very importantly, LAPLACE can be easily visualised as linear and non-linear flow-diagrams depicting exactly what the software is doing at various levels of abstraction. These visualisations can be used to construct and alter the code and can be truncated, ordered, searched and compared to show to changes structurally through both run-time and different versions. Any given process can be visualised using LAPLACE's cognitive drawing models, meaning for example instead of inputting perimeters in space, the user can point to them and then assign what they are relative to and then how they react based on other parameters. This can then be visualised in an assortment of ways including graphing and simulation. All of this leads to a great ease of debugging and behavioural correction.

Solvers

LAPINE includes a large number of solver-libraries, neural systems which are fed interacting variables and then identify what the next step is likely going to be. Eventually when enough variables through time are available, a predictive model can be created, rather than having the user guess at what this predictive model would have to be. While the solver takes a long time to execute, it takes most of the stress out of programs which are expected to react to live information. In addition, it also means the software can get smarter as the duration of available variables increases.

Structure

LAPINE is by en large a reimagining of the SCRIBOL using modern concepts and neural syntax. During its introduction it was describes simply as “SCRIBOL without the scribbling”. LAPINE does not create unsafe accessors by default (contrary to SCRIBOL), though they can be created if explicitly declared. Dot notation and plain english text (which can be parsed by language with neural weighting) can be used, meaning if an instruction can be typed in a user's language, it can be converted into running code provided the instructions are low-level enough. More importantly, the system automates closures and terminators and visualises data-structures in a way which takes advantage of object relationships and spacial awareness inherent in most species who will be using the programming language: essentially meaning a user can do a lot of the coding in diagrams.

LAPINE can either be compiled on run to match the system it is running on (in which case it is distributed as an open source application) or elements of it can be pre-compiled and run using OtherOS to emulate hardware calls and falsify operating environment conditions as a closed-source application.

Trivia

LAPINE's original designer left a number of key calls embedded in the language. The names actually reference breeds of Ey'tis (a type of Lorath rabbit which is closer in form to a wolf – Crixa, elil, embleer, flay, Frith, hlao, homba, hrair, lendri, Owsla, tharn, threar, Sayn, Threarah) as well as a number of Lorath castes which later either died out or converged into the castes known today.

Extensions

Key to LAPLACE are its modular elements. To clarify, a module is any independent transplantable unit of measurable software. While this might sound incredibly vague, extensions come in a huge number of flavours:

- Applications
- Services
- Engines
- Libraries
- Entities