ROGEX LABORATORIES

RXos: un sustrato neuromórfico verificable sobre hardware von Neumann

Cómo funciona el código, y cómo comprobar — no creer — que el comportamiento neuromórfico es real.

Paper técnico · Rev 1.0 · Roger N. alias @navywakura · Verificado en HP 15-ac195nl (i7-5500U, 8 GB) y QEMU

Qué afirma este documento y qué no. Afirma que RXos ejecuta un sustrato de eventos con dinámica de integración y disparo con fugas, en aritmética entera exacta, sobre paths vivos del sistema operativo, y que esa afirmación es *falsable* con un comando. No afirma que RXos sea hardware neuromórfico: el silicio sigue siendo x86_64 con reloj, y esa frontera se trata explícitamente en la sección 6.

1. El problema con la palabra «neuromórfico»

Es fácil dibujar una neurona. Una barra que crece, cruza un umbral, se reinicia y vuelve a empezar es igual de compatible con un contador y un módulo que con un modelo de membrana. Si el criterio para llamar a algo neuromórfico es que *parezca* una neurona en pantalla, entonces la palabra no distingue nada y no vale la pena usarla.

Este documento adopta un criterio distinto: una propiedad concreta, predecible desde el modelo, y comprobable contra el binario que arranca en la máquina. La propiedad es **codificación temporal**: que la misma entrada produzca resultados distintos según *cuándo* llegue.

El criterio, en una frase. Tres estímulos que suman 1,05× el umbral disparan la unidad si llegan juntos y no hacen nada si llegan separados. Misma entrada, misma suma, resultado distinto, decidido solo por el tiempo de llegada. Ningún acumulador, contador ni bucle sondeado tiene esa propiedad.

2. Cómo funciona el código

2.1 Del arranque al lazo de eventos

GRUB entrega el control por Multiboot2 en modo protegido. `boot/boot.asm` comprueba CPUID y modo largo, construye tablas de páginas de 4 niveles que mapean identidad los primeros 4 GiB con páginas enormes de 2 MiB, activa PAE + EFER.LME + CR0.PG, carga una GDT de 64 bits y salta a `kmain()`. A partir de ahí no se vuelve a llamar a ningún servicio de firmware.

El detalle de los 4 GiB no es cosmético: en una máquina de 8 GB la región de RAM utilizable más grande está *por encima* de 4 GiB, y el asignador la elegía. Repartía marcos en `0x1_0000_0000`, que nadie había mapeado, y la primera reserva provocaba un fallo de página antes de llegar al shell. El mapa de 4 GiB cubre además todo el hueco MMIO de 32 bits, donde una máquina real pone sus BAR de PCI, el APIC local y la apertura de la GPU integrada.

2.2 El tejido de eventos

El núcleo neuromórfico vive en `kernel/event/`. Sus piezas:

El evento

`rx_event_t` ocupa exactamente una línea de caché de 64 bytes, alineada. Lleva tipo, marca de tiempo en ticks del PIT, un valor de estímulo en Q16.16 y una carga útil. El tamaño no es estético: un evento por línea de caché significa que un productor y un consumidor no comparten línea, y que el coste de mover un evento está acotado por el subsistema de memoria y no por el compilador.

El anillo

Cada actor tiene una bandeja de entrada: un anillo SPSC (un productor, un consumidor) sin bloqueos. El productor es siempre un ISR; el consumidor es siempre el pump. La publicación ordena con `mfence`. La seguridad del contrato de productor único no se asume: las puertas de interrupción dejan IF=0, así que IRQ1 e IRQ12 no pueden anidarse y ambos pueden producir en el mismo anillo sin carrera.

La unidad LIF

Un actor es una unidad de integración y disparo con fugas. Su potencial de membrana es un entero Q16.16 — el kernel se compila con `-mno-sse` y no hay estado de FPU en ninguna parte, así que la dinámica *tiene* que ser entera. La fuga no se aplica con un tick periódico; se calcula perezosamente en el momento en que llega el siguiente evento, a partir de la diferencia de marcas de tiempo:

$$V(\Delta t) = V \cdot \tau / (\tau + \Delta t)$$

Es una aproximación racional a la exponencial, elegida porque se evalúa exactamente en enteros de 64 bits sin tablas ni series. Que sea una aproximación se dice aquí y se comprueba en la sección 3: el paper no publica la ecuación canónica mientras el kernel evalúa otra cosa.

No hay tick neural en ningún sitio del kernel. Esa es la diferencia estructural con una simulación: un simulador de redes de impulsos avanza el tiempo en pasos y evalúa cada neurona en cada paso. Aquí no existe ese bucle. Un actor que no recibe eventos no consume ni un ciclo, y su membrana sigue siendo correcta cuando por fin llega uno, porque la fuga se reconstruye desde la marca de tiempo.

La sinapsis

Cada actor lleva un registro de plasticidad local: el peso se mueve según el orden relativo entre el evento presináptico y el disparo postsináptico (STDP), también en Q16.16.

2.3 Dónde está el sustrato en caminos vivos

Un tejido de eventos que solo alimentase una demo sería un juguete. Estos son los caminos reales del sistema operativo que pasan por él:

CAMINO	CÓMO USA EL TEJIDO
Teclado y ratón	IRQ1 e IRQ12 producen descriptores `RX_EVENT_KBD` / `RX_EVENT_MOUSE` en el anillo SPSC. **Nunca se umbralizan**: la entrada es determinista por diseño, y esa es la frontera crítica.
Recepción de red	El actor de NIC recibe eventos de aviso INTx y un estímulo de plazo de 1 tick. La política es de disparo siempre y determinista, porque la telemetría demostró 0 entregas INTx en q35: umbralizar el drenaje habría repetido una regresión anterior. Se documenta como decisión medida.

Refresco del gestor de ventanas	El primer actor genuinamente umbralizado. Sustituye al repintado por temporizador a 1 Hz: solo recibe estímulo mientras hay una ventana de telemetría en pantalla, así que un escritorio estático no repinta nada.
Lazo principal	La inversión está hecha: `kmain` lanza el shell como tarea con `sched_spawn` y entrega la máquina a `rx_kernel_event_loop()`, que mueve cada pump, cede a las tareas listas y aparca el núcleo cuando nadie tiene trabajo.

2.4 El planificador y la energía

El cambio de contexto cooperativo guarda el conjunto que exige la ABI System V (rbx, rbp, r12–r15) más rsp. No hay marco de interrupción, ni estado de FPU, ni recarga de segmentos, porque nunca se sale del anillo 0. No es apropiativo a propósito: un cambio dirigido por temporizador introduciría reentrada en drivers escritos monohilo.

Al quedarse sin trabajo, `power_idle()` aparca el núcleo en el estado más profundo que la máquina soporta de verdad: MONITOR/MWAIT con una pista de estado C cuando CPUID lo expone, y HLT en caso contrario. Y `power` mide el consumo real con los contadores RAPL, leyéndolos con una recuperación de #GP para poder preguntarle a cualquier CPU sin arriesgar un pánico de arranque.

3. Cómo comprobarlo: el comando `bench`

El comando `neuro` muestra una neurona en marcha. Eso es una demostración. `bench` es la otra cosa: experimentos cuyo resultado se **predice desde el modelo primero** y luego se contrasta con el kernel en ejecución. Cada fila dice qué exige el modelo, qué dio el kernel, y si coinciden. Una discrepancia es un fallo, no una nota al pie.

```
test                     model requires  kernel gave
-----------------------------------------------------------
temporal: 3 stim, fast   1 spike         1 spike        PASS
temporal: 3 stim, slow   0 spikes        0 spikes       PASS
decay vs V*t/(t+dt)      5/5 exact       5/5 exact      PASS
determinism, 72 steps    bit-exact       bit-exact      PASS
refractory lockout       0 then 1        0 then 1       PASS
sparsity: 200 sub-thr    200 absorbed    200 absorbed   PASS

cost per event   299 cycles (integer only: the kernel is
                 built -mno-sse, there is no FPU state)

VERDICT: 6/6 PASS
```

Todo corre en **tiempo virtual**: los valores de tick se le pasan al actor explícitamente en vez de leerlos del PIT. El resultado es por tanto idéntico en un portátil de 2,4 GHz y dentro de un emulador, reproducible ejecución tras ejecución, y no una medida de lo rápido que resulta ser esta máquina. El único número de reloj de pared —ciclos por evento— va etiquetado como tal.

3.1 Qué demuestra cada fila

FILA	QUÉ DESCARTA
Codificación temporal (rápido)	Suman 1,05× el umbral y la fuga apenas los toca: el tercero cruza. Confirma que integra.

3 estímulos en ticks consecutivos	
Codificación temporal (lento) los mismos 3, separados 2τ	**La fila decisiva.** Cada uno ha decaído a un tercio antes de que llegue el siguiente, y la membrana nunca llega. Un contador habría disparado igual. Descarta acumulador, contador y bucle sondeado de un solo golpe.
Fuga contra el modelo 5 puntos de Δt	La columna esperada es aritmética hecha a mano, no una segunda llamada al mismo código, así que la coincidencia es evidencia sobre la implementación y no una tautología. Descarta que la fuga sea decorativa.
Determinismo 2 actores, 72 pasos	Trazas idénticas bit a bit. Una implementación en coma flotante sería libre de no serlo. Confirma que la dinámica es entera y exacta.
Bloqueo refractario	Un estímulo de 2× el umbral dentro de la ventana se absorbe; el mismo estímulo fuera dispara. Descarta que el umbral sea un simple `if`.
Dispersión 200 estímulos subumbral	Los 200 se integran y los 200 se absorben: la retrollamada no se ejecuta ni una vez, así que no se hace *ningún* trabajo aguas abajo. Esa proporción es la afirmación económica de un sustrato dirigido por eventos, y aquí se cuenta en vez de afirmarse.

> **El benchmark encontró dos bugs reales el día que se escribió.** La fila de la fuga discrepaba del modelo en todos los Δt salvo cero: la membrana no decaía nunca para un actor cuya primera actividad caía en el tick 0, porque el código comprobaba `last_update_ticks != 0` y confundía una marca de tiempo válida de cero con «nunca actualizado». Al corregirlo apareció el segundo: el Makefile no rastreaba dependencias de cabeceras, así que añadir un campo a la estructura no recompilaba nada y medio sistema quedaba con el layout antiguo — un #GP con un `rip` basura y ningún error de compilación. Esa es la utilidad de un benchmark que predice antes de medir.

4. Hechos medidos

MAGNITUD	VALOR	CÓMO SE OBTIENE
Filas del benchmark que pasan	6 / 6	`bench`
Coste por evento	~299 ciclos	`bench`, RDTSC sobre 20 000 estímulos
Aritmética de la membrana	Q16.16 entera	el kernel se compila `-mno-sse`
Tamaño del evento	64 B, alineado	`rx_event.h`
Ticks neurales periódicos	0	no existe tal bucle en el árbol
RAM en uso al arrancar	~3 MiB de 8191 MiB	`mem`
Auto-test del tejido	PASS en cada arranque	línea de arranque
Suites automatizadas	26 + 21 + red	`make test`, `test-disk`, `net-test`

5. Estado en hardware real

La máquina de referencia es un HP 15-ac195nl: Core i7-5500U (Broadwell), 8 GB DDR3L, panel 1366×768, SSD de 476 GiB. Arranca por Legacy/CSM, negocia el modo de vídeo con el panel, y con el asistente `install` escribe una MBR con una partición RXos propia (tipo `0x7F` desde el LBA 2048), formatea RXFS dentro y guarda el perfil. El siguiente arranque dice **«Welcome back»** y restaura los archivos. Verificado por el autor sobre la máquina física.

Lo que ese hardware añade a este documento es el comando `power`: los contadores RAPL no los implementa ningún emulador, así que la única cifra del proyecto que **solo** puede producirse en metal es el consumo medido. La instrumentación está verificada; el número concreto corresponde publicarlo a quien tenga la máquina delante, y este documento no lo inventa.

6. Qué significa «neuromórfico sobre von Neumann» y qué no

Conviene ser exacto, porque aquí es donde se rompen la mayoría de las afirmaciones de este campo.

Lo que RXos sí es

- Un sustrato **dirigido por eventos**: sin tick neural, sin sondeo, sin trabajo cuando no hay eventos.
- Con **dinámica temporal real**: el resultado depende de cuándo llegan los estímulos, no solo de cuántos.
- **Determinista y exacto**: enteros, sin coma flotante, reproducible bit a bit.
- Con **plasticidad local**: STDP por sinapsis, sin regla global.
- **Sobre caminos vivos** del sistema operativo, no en una caja de arena.

Lo que RXos no es

- **No es hardware neuromórfico.** El silicio tiene reloj, la memoria y el cómputo están separados, y cada evento cuesta ciclos de CPU. El cuello de botella de von Neumann sigue ahí, intacto.
- **No es masivamente paralelo.** Los actores se despachan en serie desde un pump cooperativo. Un chip neuromórfico evalúa millones de unidades a la vez; aquí se evalúan una detrás de otra.
- **No es asíncrono.** El tiempo se mide en ticks del PIT a 100 Hz. La resolución temporal del sustrato es 10 ms, no nanosegundos.
- **La tolerancia a fallos es redundancia clásica** con vocabulario neuromórfico. Sin un crossbar físico no hay reencaminamiento real, y el código lo etiqueta como lo que es.

> **Qué falsaría la afirmación.** Que una fila de `bench` fallara y no se explicara. Que apareciera un bucle que evalúe actores periódicamente. Que la dinámica dependiera del reloj de pared en vez de las marcas de tiempo de los eventos. Que la fila «lento» de codificación temporal disparara. Cualquiera de las cuatro cosas se comprueba con un comando y termina el debate.

7. Reproducirlo

```
# en la máquina, desde el shell o desde la terminal del WM (tecla t)
bench          # los seis experimentos, modelo contra kernel
neuro          # la neurona en vivo movida por tus pulsaciones
status         # contadores del tejido: eventos, descartes, tareas
mem            # uso real de memoria, no "libre / total"
power 5        # vatios medidos - solo en metal

# en el equipo de desarrollo
make test        # 26 comprobaciones de humo en QEMU
make test-disk   # 21, persistencia en tres backends
make net-test    # ping de dos nodos por Ethernet crudo
```
