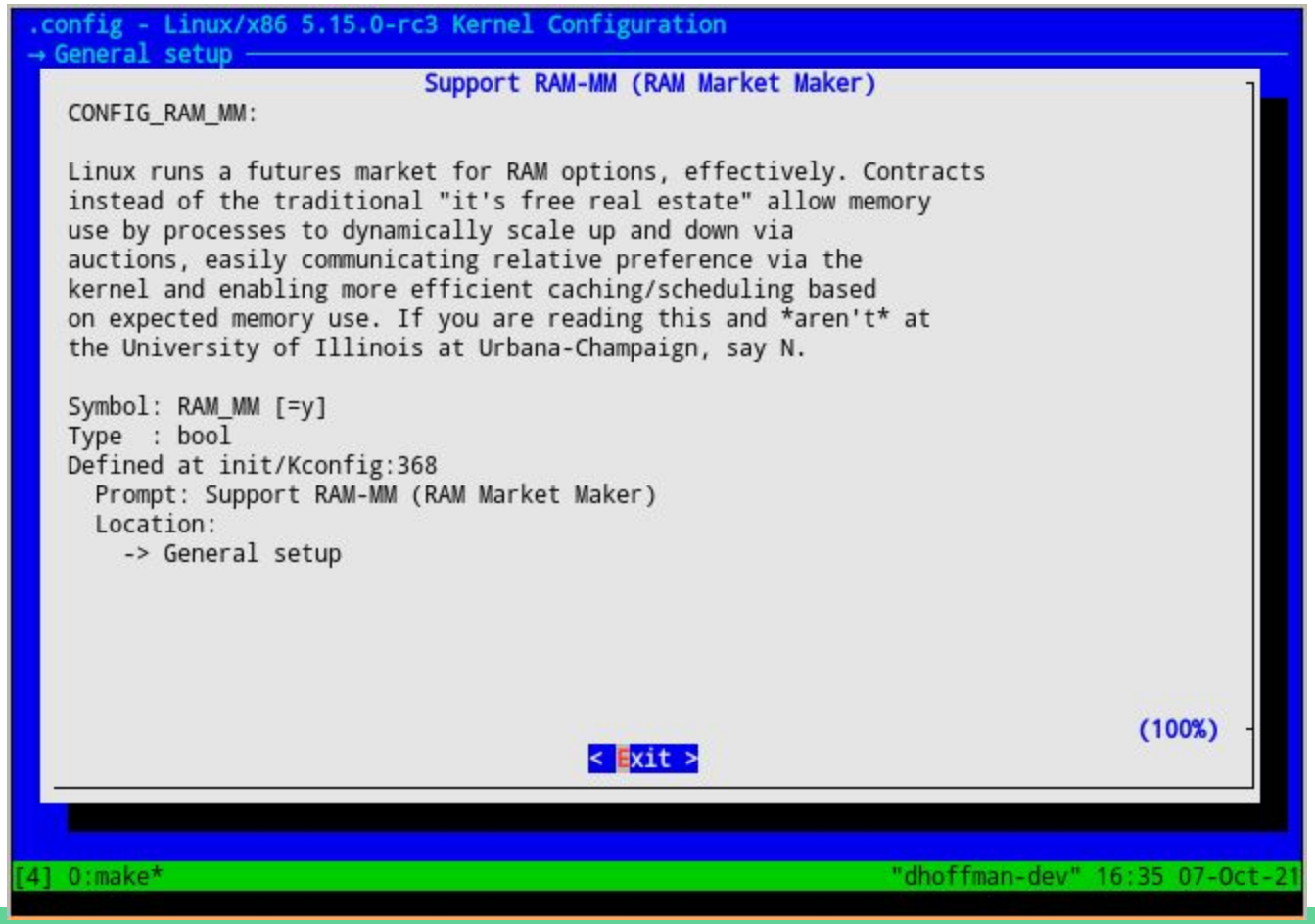
Linux RAM-MM

Daniel Hoffman

What?


.config - Linux/x86 5.15.0-rc3 Kernel Configuration
→ General setup
Support RAM-MM (RAM Market Maker)
CONFIG_RAM_MM:
Linux runs a futures market for RAM options, effectively. Contracts
instead of the traditional "it's free real estate" allow memory
use by processes to dynamically scale up and down via
auctions, easily communicating relative preference via the
kernel and enabling more efficient caching/scheduling based
on expected memory use. If you are reading this and *aren't* at
the University of Illinois at Urbana-Champaign, say N.
Symbol: RAM_MM [=y]
Type : bool
Defined at init/Kconfig:368
Prompt: Support RAM-MM (RAM Market Maker)
Location:
-> General setup
(100%)
< Exit >
[4] 0:make*
"dhoffman-dev" 16:35 07-Oct-21

How does it work?

- Userspace
  - `malloc` with price information -> `mmap` with price information -> syscall
  - Create an interrupt handler on `SIGUSR1` to handle eviction requests
- Kernelspace
  - Price information is bound to memory mapped page
  - Kthread runs in the background (`krammmd`)
    - Calculate what, if anything, should be evicted, register with a linked list
    - Send an interrupt to the process
    - Process reads pointer from linked list via `prctl` then calls `munmap`
  - Kernel also tries to evict pages when it needs RAM *now*, but its too late to wait for userspace

Market Design Considerations

- Does the Linux kernel exercise any preference at all?
  - De-duplicated pages can’t be freed unless the last virtual memory reference is free’d
    - It works like a garbage collector
  - Easily compressible pages (and compressed) pages use less space than non-compressed pages
- The concept of a RAM credit score
  - The only universal way to free RAM of a process is via SIGKILL or panicking the kernel
  - How well does the program estimate its own memory requirements?
  - How long does it take to respond to interrupts?
  - How does this information persist over time? We probably need a userspace daemon too...
- How do we incentivize RAM contracts while maintaining backwards compatibility?

Implementation Design Considerations

- How would `malloc` work?
  - To the user, RAM is allocated in size passed into `malloc`, allocation and free is done as pages
  - Co-locate similar contracts to maximize evict-ability since we can't change RAM addresses after the fact
  - Another approach could be to have the kernel directly write memory changes
    - We require a standard `malloc` representation
    - Pass pointers via syscalls to register our memory allocator information with the kernel
    - Directly evict without context switching means more flexibility (99.9% usage all the time and evict elastic pages in the OOM)
- How can this work with KVM/hypervisor solutions?
  - Yo dawg I heard you like VMs so I QEMU'd your ESXi so you can KVM while you LXC
- There are plenty of other considerations entirely within the Linux kernel, but I've only listed those that change userspace

Practical Use Cases

- Very small: Microcontrollers
  - Microcontrollers don’t typically run Linux (no MMU), but similar concepts can probably be generalized through something like FreeRTOS
- Very large: virtualization hypervisors (Proxmox, ESXi, etc)
  - RAM pressure can come from the outside
  - Most machines assume they can use 100% of the RAM allocated to them, even in VMs

Status

- What works
  - `malloc` -> `mmap` -> syscall communicates price
  - Price information is stored alongside the memory mapped page
  - `krammmd` evicts RAM pages based on exercised preference, fires interrupt
  - Interrupt is handed, mmap'd address is read from another syscall,`munmap` is called
- What needs improvement
  - `krammmd` is *very* naive and uses some magic numbers, *very* slow
  - Reasonable `malloc` re-implementation
    - Co-locates similar contracts for more efficient evictions
    - Maybe allow for direct kernel writes for evictions (much faster)
  - Reasonable userspace implementations of RAM evictions (and incentives for participation)
    - If we opt for microcontroller applications, then we can develop a closed system faster
    - If we opt for desktop applications, then there is more practical benefit to larger adoption

Future Ideas

- Register `malloc` implementation with the kernel so it can evict pages itself
  - This exists *alongside* the interrupt system
  - Useful for guaranteed performance since there are no context switches
- Perhaps we can modify POSIX timers too...
  - Pre-negotiate RAM contracts per-execution, +/- time tolerance for execution
- Software floating point in kernelspace doesn't appear to exist...
- Create a custom kernel allocator (similar to SLAB)
- Infrastructure TODOs
  - Benchmark over time
  - Automated testing
    - X86 and ARM in QEMU
  - Automated deployment to some shitty computer
    - ACM has 256MB RasPis (no wifi), RasPi Zeroes (but that's 1GB)
    - Probably best to auto-deploy to Proxmox but a physical presence is always nice
  - Get a real Proxmox computer in ACM
    - SIGECOM Proxmox is running in a VM on my own Proxmox
    - Its a bit slow and likes to run out of RAM (ironic)

Questions?