In this essay, I will discuss the technical design of paging within the ARMv8-A architecture. It is relevant to note that paging in ARM's documentation is referred to as "translation", and as such page tables as "translation tables".

The ARMv8-A architecture enables processors to run in one of two execution states at a time: AArch64, and AArch32. The AArch64 state allocates programs 64-bit virtual addresses, whereas AArch32 allocates programs 32-bit virtual addresses. This enables backwards compatibility for 32-bit programs written for the previous 32-bit ARMv7-A architecture. For the remainder of this essay, unless specified, the translation table design will focus on the AArch64 execution state, as AArch32 state is almost identical to the design of the ARMv7-A architecture.

Virtual memory addresses are split into two sections. The first is allocated to applications, and the other is allocated to the OS and peripherals. The two sections are translated by their own translation table: TTBR0_ELn, and TTBR1_EL1 respectively[1]. There is also a region of virtual memory separating the two mapped sections. This region is unmapped, and so addresses are unaccessible, leading to a fault when addressed.

The size of the two translation tables is configurable, but has a maximum size of 48-bits of address space. This is because the 64-bit logical addresses

As we have a 2-level table design, we

0.1 Translation Tables Structure

0.2 TLB (Translation Look-aside Buffer)

0.3 Tagging

0.4 Caching

0.5 Sharability

how schedulers can be improved.

My round-robin design set a time-quantum, which would limit the amount of time the scheduler will wait for a particular process (in milliseconds). This prevents processes which are particularly intesive from blocking the other processes entirely, which enable the less intensive processes to complete earlier, which reduces the total wait time of the scheduler.

My design of this round-robin approach forked a child which ran, instead of just waiting in the parent

[1]This is used to restrict access to OS and peripheral address space as TTBR1_EL1 requires a higher privilege level, thus requiring applications to ask the OS for access