Design Issues in Managing Main Memory

I will use a metaphor of a 'processing factory' to discuss the design issues surrounding main memory management. At this factory, tasks (instructions and data) brought in from users get packaged into envelopes and become processes for the processing unit (CPU) to execute. The CPU only has a few shelves (registers) for the envelopes so there is a separate building on site called main memory (MM) to store envelopes (processes) ready to execute. As this is a large factory, it needs its own processes and staff to help run it; essentially it has an operating system (OS). Running the factory poses challenges: **Building space is limited & expensive**, it **takes staff time** to ferry the envelopes around the site, and most of the time, **there is a queue** of trucks full of user's tasks waiting to get into the factory. So, the factory needs to be optimally designed to **maximise efficiency** (minimise time & resources needed to transfer envelopes around) and **utilise building space** well (minimal empty shelves), requiring optimisation of the factory (hardware) and the system (OS).

To assist with this**, the shelves in MM should have addresses**, and **envelopes should be labelled with the address** corresponding to its address in MM. To avoid confusion though, the envelopes for the **OS tasks and the user tasks need to be kept separate** in the MM building, otherwise mix-ups and mistakes in the CPU can happen, leading to user dissatisfaction or even halting of the factory. So, **MM should be split up into separate sections** for OS and user tasks. In fact, we split it up into multiple sections so we can get lots of envelopes in and out and easily see where the free space is.

There is another logistical challenge posed by the addressing system: **tasks that arrive from users rarely state where they are destined for in MM** (they don't come with a correct MM address as users don't know the factory's addressing system). It is therefore up to the factory to allocate. So, each task that is loaded into an envelope (process) by the CPU gets a stamp from the CPU with an address. Before going to MM though, **the CPU should also ensure the envelope is allowed to go to the address it has been given** (remember we can't mix certain envelopes in MM). The address is checked against an 'allowed' base and limit address stored on the CPU's relocation shelf for this process, and given a **seal of validity**. Envelopes can either go to MM or, for efficiency, be stored back in the trucks (disk) and **dynamically swapped in and out** of the MM when needed. When the factory's really full, there's also a 'backing store' in the car park for overflow from MM, but it's not used often. The factory also requests its users include a small reference (a stub) to any commonly used tasks at the factory, rather than copying these instructions in their own tasks – the factory can cleverly **links the common tasks** to theirs at execution.

The addressing system is good, but what happens if an envelope needs to move locations in MM during its execution (if the task involves using different locations in MM). In this case the addresses in MM are going to be different to the address stamped on the envelope by the CPU. A very smart factory worker suggests a solution: have a building in between the CPU and MM called the **main memory unit (MMU)** to intercept envelopes during execution and **dynamically correlate (map) their CPU address (logical address) to the real location (physical address)** needed in MM.

Wow, our factory is really high tech now, it must be working efficiently? No! A worker informs us that there is a problem in MM – sometimes the envelopes don't fill up all their allocated section (wasted space!) and sometimes they can see lots of free space scattered but not together in a block to allow an envelope to fit. She's heard about this problem at another factory: it's called **fragmentation**. She relays some solutions stolen from the other factory. We can break the processes (envelopes) up and allow them to go to **space anywhere that's available in MM, not just space that's all free together.** One way to do this would be to divide the CPU(logical) and MM into blocks and map between them by **looking up indexes of available MM blocks in a table (page table)**. So, envelopes would be stamped with indexes instead of addresses. If the page table gets too big to store in the factory, it could be split up further into **multiple page tables** or, a faster/smaller store of common page table contents could be made next to the MMU. We could also **map backwards** from MM to logical so there is only one logical entry in the page table per physical. Or another table **(hash table)** could be used to help with the looking up of logical addresses: logical addresses (elements) point to each other and are looked up sequentially until a match is found. Sometimes our users, helpfully, suggest logical ways to split up their tasks for them. The factory could use these suggestions and split up the envelopes into smaller **segments** and use a table to **map all the segments to a 2- dimensional physical address space**.

Citation

Silberschatz, Operating Systems Concepts 8th Edition (Chapter 8) John Wiley & Sons, 2009