VeloMax Motors Proposed Operating System for an intelligent, connected vehicle

u5539006

This poster covers the fundamentals of the operating system (OS) designed for a vehicle's Infotainment System for VeloMax Motors including scheduling, as well as process and memory management. It is important that the OS meets the needs of the user as best it can, while still being 'Secure by Design': "technology products are built in a way that reasonably protects against malicious cyber actors successfully gaining access to devices, data, and connected infrastructure." [1]

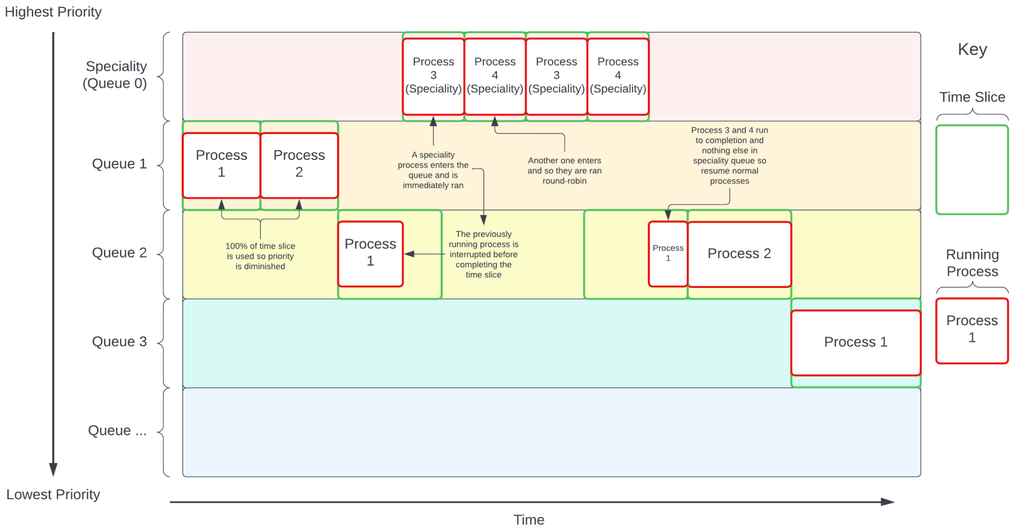

Figure 1. A diagram illustrating the "Multi-Level Feedback Specialisation Queue", following example processes during runtime

This OS uses the Multi-Level Feedback Queue (MLFQ) as the basis for its scheduling algorithm. This allows it to prioritise 'I/O-bound' tasks ("[those] that spend a significant amount of time waiting for I/O requests to complete" [2]). This includes receiving Bluetooth streams from a mobile device or reading sensory data about the vehicle.

I propose taking further factors than just runtime into account when determining a process's priority, but also using data about how much the user uses that application over the past month and giving each core (on a multi-core processor) a "specialisation" – an application that they prioritise, over everything else, processes for.

This functions similarly to interrupts on a CPU, except that each thread by a certain process would operate in a round-robin (assuming they weren't distributed to other cores) rather than consulting a priority table to see which is the most important. It is important to prioritise applications that the user uses often, so the system feels responsive to the user, but also to be able to run processes by certain applications at any time for systems that are important to the user, such as navigation, which would need a low turnaround time. Other examples of specialisation include car controls and crash alert system, music, and mobile device connectivity.

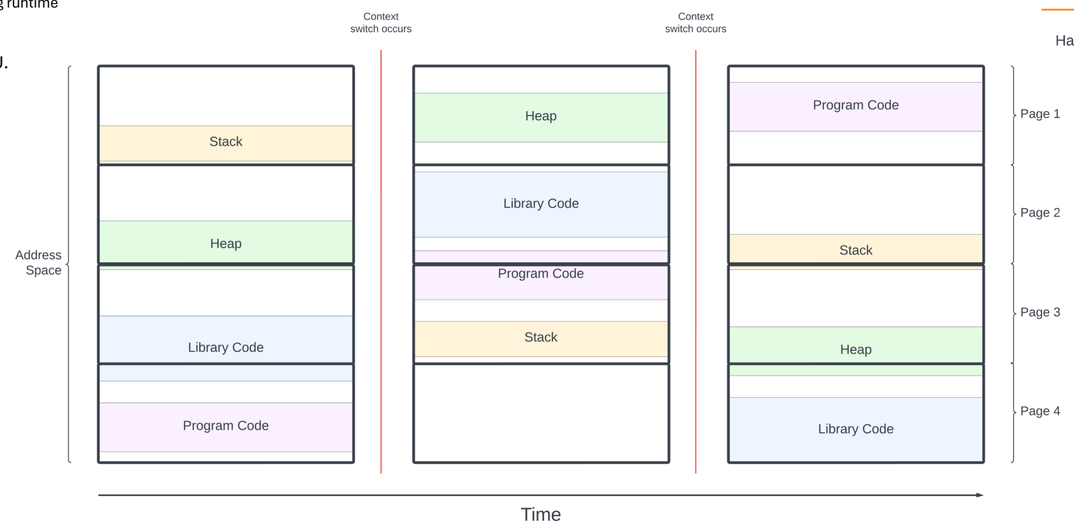

Figure 2. A diagram illustrating the randomisation of sections of a process's address space for each time a context switch occurs, swapping it to the running process

This variation of the MLFQ stops non-critical processes from monopolising the CPU.

This specialisation mechanism allows for processes to likely run on "their" core. This will speed up the process since its cache working set is already present (process affinity is likely to be high). When processes need to communicate, such as when applications need to display new data, indirect communication will allow for a more secure system, as the OS handles the delivery of such messages, and no process accesses another's data. This system utilises the message-passing principle as it produces a lower overhead and is a more flexible approach than other principles. The OS will also operate Limited Direct Execution so the OS controls sensitive functions.

Applications must obtain Mutual Exclusion Objects to access shared files which prevents race conditions from arising and causing unpredictable events to occur. In case unpredictable events occur, incremental process checkpointing takes place, so it is possible to restore a consistent version of the process and execute it again to hopefully avoid further exceptions and errors. Only changes to processes are stored since this will happen often. As a further security measure, Control Flow Enforcement will be implemented as part of the OS, to mitigate buffer overflow and code injection attacks.

To prevent deadlock, the Banker's algorithm is used to "[analyse] the currently allocated resources and the resources required by [a process] in the future". [3]

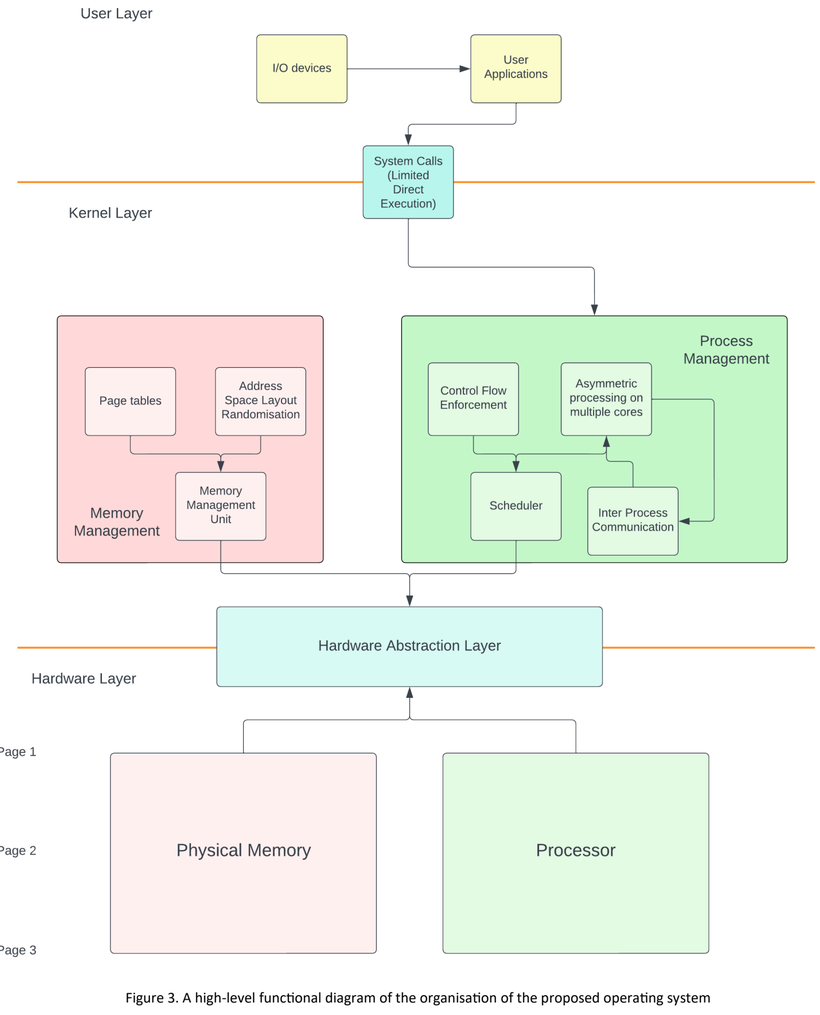

Figure 3. A high-level functional diagram of the organisation of the proposed operating system

It is vital, at every part of the system, to ensure maximum security is provided, whilst not affecting the user experience. The MLFQ allows this to happen by prioritising common user functions and allowing critical processes to be run before any other tasks. Paging allows for efficient use of memory, and ASLR helps to prevent simple position-based exploits. Process checkpointing allows processes to be re-ran, should they throw errors or exceptions.

The physical memory of this device is split into pages and requires address translation from the process's page table. Memory must be virtualised since multiple processes are running simultaneously. Pages are allocated to processes on a first-fit basis to reduce fragmentation by storing the data from a process as closely as possible. By using paging as the management principle, we also avoid polluting the beginning of memory with small objects (which happens when using first-fit with variable-size chunks). This should cause memory allocation requests to be fulfilled more quickly.

It is important when fetching data from memory to check that the data a process is requesting is within the bounds of its pages, so it doesn't end up accessing another process's data. It is also necessary to ensure that data is validated before being written to memory so that it doesn't overwrite memory owned by other processes, or the OS itself. For example, only copying data that will fill the buffer, and ignoring anything subsequent, prevents buffer overflow attacks.

Address Space Layout Randomisation (ASLR) is also implemented to help "increase the difficulty of buffer overflow attacks" [4] by ensuring that the locations of executables and libraries in a process's address space aren't standardised. This helps mitigate attacks based on return-oriented programming, and it increases the complexity of any attack on the system.

[1] *Secure-by-design: CISA* (2023) *Cybersecurity and Infrastructure Security Agency CISA*. Available at: https://www.cisa.gov/resources-tools/resources/secure-by-design (Accessed: 05 December 2023).

[2] Dosunmu, O. (2023) *Understanding I/O-bound and CPU-bound tasks in backend development*, *LinkedIn*. Available at: https://www.linkedin.com/pulse/understanding-io-bound-cpu-bound-tasks-backend-oluwamuyiwa-dosunmu/ (Accessed: 05 December 2023).

[3] *What is Banker's algorithm?* (2020) *AfterAcademy*. Available at: https://afteracademy.com/blog/what-is-bankers-algorithm/ (Accessed: 05 December 2023).

[4] (2023) *Address space layout randomization*. Available at: https://www.ibm.com/docs/en/zos/3.1.0?topic=overview-address-space-layout-randomization (Accessed: 05 December 2023).