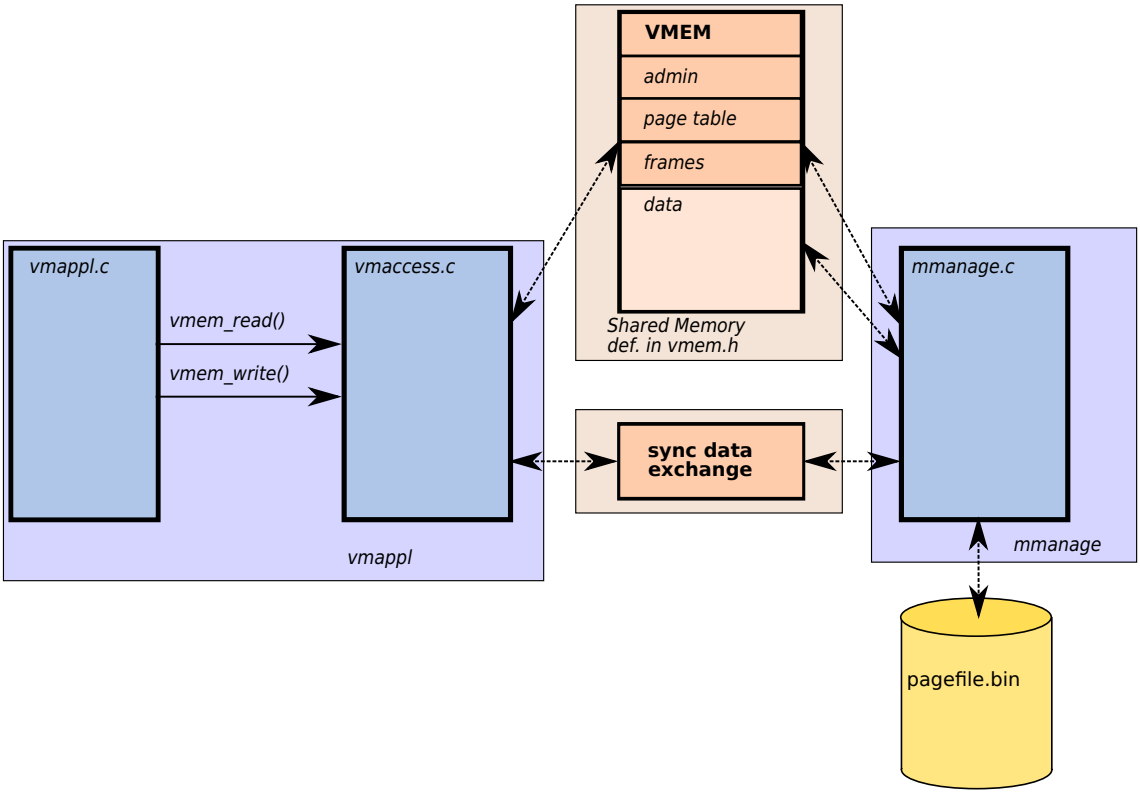
3 Virtuelle Speicherverwaltung

Diese Aufgabe untersucht Mechanismen der virtuellen Speicherverwaltung. Es wird eine Simulation für die Seitenersetzungsalgorithmen FIFO, CLOCK und AGING erstellt. An die Stelle des physikalischen Speichers eines realen Systems tritt hier ein Speicherbereich im *Shared Memory*. Über einen weiteren gemeinsamen Speicher sendet die Anwendung Aufträge an den Memory Manager (`syncdataexchange.c`). Über zwei Semaphore wird der Datenaustausch als synchrone Kommunikation realisiert.



Komponenten der Anwendung:

vmappl.c Dieses Modul stellt das Anwendungsprogramm dar. Es werden zufällige Daten erzeugt, angezeigt, mit Quicksort oder Bubblesort sortiert und erneut angezeigt. Der Parameter *-seed* initialisiert den Zufallszahlengenerator.
Den Quellcode finden Sie in meinem Pub-Verzeichnis. Ändern Sie `vmappl.c` nicht, damit die Anzahl der Seitenfehler Ihrer Speicherverwaltung identisch mit dem Resultat der Musterlösung ist.

vmaccess.c bildet die Schnittstelle zum virtuellen Speicher. Im realen System ist dies die Adress-Dekodierungseinheit (Umsetzung von virtuellen auf reale Adressen). Die Methoden `vm_read` und `vm_write` berechnen aus der virtuellen Speicheradresse die Frame-Nummer und den Offset.
Ist die benötigte Seite nicht geladen, wird die Speicherverwaltung (`mmanage.c`) über einen Auftrage (`syncdataexchange.c`) aufgefordert, die Seite aus dem File `pagefile.bin` in den Hauptspeicher zu laden. Sie sucht einen freien Seitenrahmen, lagert ggf. eine Seite entsprechend den Algorithmen FIFO, CLOCK oder

AGING aus und liest die gewünschte Seite ein.
Die Routinen aus `vmaccess.c` blockieren so lange bis `mmanage` die synchrone Kommunikation mit einem ACK abgeschlossen hat.

mmanage.c Dies ist die Verwaltung der im Hauptspeicher vorhandenen Seitenrahmen. Nach der Initialisierung warte das Modul auf Aufträge von `vmappl`.
Den Zugriff auf die `pagefile` Datei realisiert das Modul `pagefile`. Unter in meinem Pub-Verzeichnis finden Sie eine Implementierung des Moduls.
Beim Start initialisiert `mmanage.c` das Shared Memory, die synchrone Kommunikation, installiert mit `sigaction()` die Signalhandler, erzeugt bei Bedarf die Datei pagefile.bin und initialisiert die Datenstrukturen. Ein Teil dieser Funktionen finden Sie im Module `mmanage.c` fertig implementiert.
Außerdem ruft `mmanage.c` bei jedem Seitenfehler die Methode `logger()` auf, die die Aktion im Logfile `logfile.txt` protokolliert. Diese Methode sollten Sie nicht ändern. So kann man Logfiles mit `diff` vergleichen. Ein entsprechendes Modul finden Sie im mitgelieferten Quellcode.
Beendet wird das Programm mit <Strg>-<C> (also über das Signal SIGINT). Alle Ressourcen müssen dem Betriebssystem zurück gegeben werden. Sonst kann es zu Problemen bei einer erneuten Ausführung der Simulation kommen. Also müssen Sie die entsprechenden cleanup Funktionen im Signalhandler zu `SIGINT` einhängen. Die mitgelieferte Datei `mmanage.c` enthält Teile des notwendigen Quellcodes.

vmem.h Hier wird die Datenstruktur **`struct`** `vmem_struct` mit allen weiteren Strukturen und Konstanten definiert.

- Bitte arbeiten Sie vorab die mitgelieferte Dokumentation dieser Datenstrukturen durch.

Aufgabe:

- Schreiben Sie gemäß der oben spezifizierten Anwendung die Speicherverwaltung `mmanage`. Der Code des „Anwendungsprogramms“ `vmappl` finden Sie in meinem Pub-Verzeichnis.
  Dort finden Sie zu Ihrer Unterstützung finden Sie neben dem Code des Anwendungsprogramms einige Hilfsfunktionen und Datenstrukturen. Auch die entsprechende DoxyGen Dokumentation finden Sie dort.
- Außerdem finden Sie das Shell Skript `run_all`. Es führt mehrere Simulationen durch und vergleicht die Simulationsergebnisse gegen die Musterlösung (via diff). Das Skript führt die Simulation mit unterschiedlichen Sequenzen von Pseudozufallszahlen durch. Ein Vergleich gegen die Musterlösung findet nur für den seed 2806 statt. **Verwenden Sie in der Testphase nur den seed 2806 - das spart Rechenzeit. Dazu muss eine Variable im Skript angepasst werden.**
- Ein `Makefile` finden Sie im Code im Pub-Verzeichnis. **Dieses Makefile müssen Sie im Rahmen der Abnahme erklären können.**

- `run_all` erzeugt zu FIFO, CLOCK und AGING Varianten für Seitengrößen von 8, 16, 32 und 64 Datenworten (**`int`**).
  AGING benötigt ein Zeitintervall zum Abfragen der R-Flags. Dies wird durch den globalen Zähler `g_count` simuliert, der bei jedem Speicherzugriff inkrementiert wird.
- Gemäß dem Shell Skript `run_all` wird die Simulation für die Sortieralgorithmen BubbleSort und QuickSort durchgeführt. Weiterhin wird das zu sortierende Feld mit unterschiedlichen Werten initialisiert (s. -seed Parameter von `vmappl`). `run_all` erstellt eine Tabelle, die für unterschiedliche `seed`s die Anzahl der Seitenfehler mit den grundlegenden Parametern der virtuellen Speicherverwaltung in Verbindung setzt. Da eine Simulation, deren Ergebnisse nicht interpretiert werden, sinnlos ist, interpretieren Sie bitte die Tabelle / den Graph.
- Hilfsfunktionen zum einheitlichen Melden von Fehlern finden Sie in `debug.h`.

Abnahme der Aufgabe:

Im Pub-Verzeichnis finden Sie die Log Files für die drei Seitenersetzungsalgorithmen, wobei der Zufallszahlengenerator mit 2806 initialisiert wurde. **Im Rahmen der Abnahme wird überprüft, dass Ihre Lösung die selben Logfiles generiert.** Es ist wichtig, dass Sie schon vor dem Praktikumstermin sicherstellen, dass Ihre Logfiles mit meinen übereinstimmen. Beachten Sie folgende Punkte:

- Die Log Files für die Seitenersetzungsalgorithmen FIFO und CLOCK sollten schnell mit Ihrer Lösung übereinstimmen. Erst wenn diese stabil laufen, sollten Sie das AGING-Verfahren umsetzen.
- Falls die Log Files zu Ihrem AGING-Verfahren nicht mit den gegebenen Logfiles übereinstimmen, dann müssen Sie anhand der Ausgaben Ihres Programms die korrekte Funktionsweise des AGING-Algorithmus darstellen können.
- Die Musterlösung und der Code im Pub-Verzeichnis können fehlerhaft sein. Wenn Sie einen Fehler finden, schicken Sie mit bitte eine E-Mail. Erklären Sie kurz den Fehler.