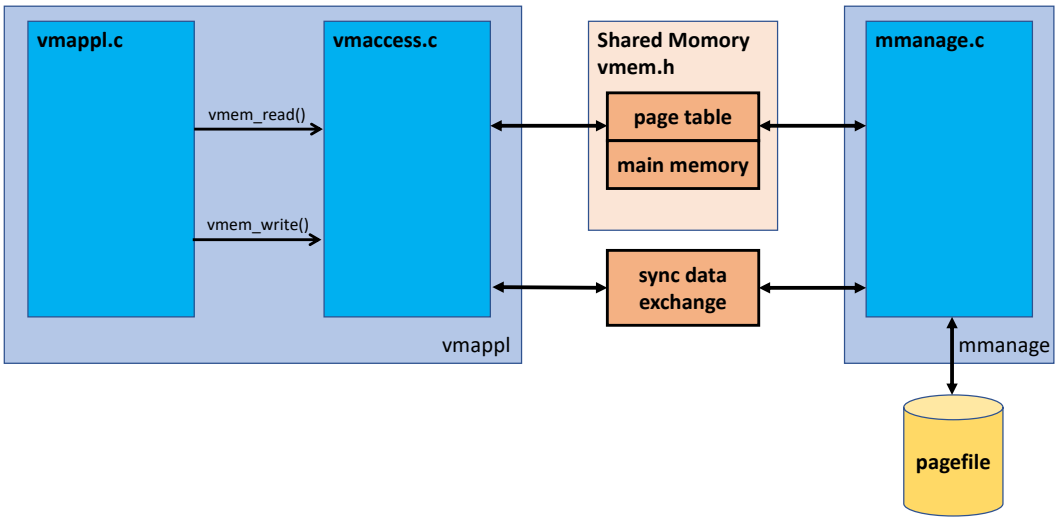
3 Virtuelle Speicherverwaltung

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.
Neben der Lösung der Aufgabe selbst gibt es die weitere realitätsnahe Anforderung, dass Sie existierenden Code anpassen müssen. **Bitte arbeiten Sie folgende kleine Einführung durch. Lesen Sie anschließend die Dokumentation des Codes und gleichen Sie diese mit der Einführung ab.** Das wird Ihnen viel Zeit ersparen.



- Die obige Abbildung stellt die beiden Prozesse `vmappl` und `mmanage` dar. `vmappl` ist eine Anwendung, die ein Feld von ganzen Zahlen wahlweise mit QuickSort oder BubbleSort sortiert. `mmanage` implementiert die virtuelle Speicherverwaltung.
- Die Prozesse kommunizieren über zwei Shared Memory Bereiche. Der `SHMKEY` des ersten Bereichs ist `./src/vmem.h` . Die zugehörige C Datei lautet `vmem.h` . In diesem Bereich liegt zum einen das `mainMemory` . Es repräsentiert den Hauptspeicher und ist `VMEM_PHYSMEMSIZE` Byte groß. Der Hauptspeicher ist in Seitenrahmen der Größe `VMEM_PAGESIZE` unterteilt.
  In der Realität liegt der gesamte virtuelle Speicher auf der Festplatte. Dies wird durch die Datei `pagefile` nachgebildet. Die Größe des virtuellen Speichers beträgt `VMEM_VIRTMEMSIZE` Byte. Er ist in Seiten der Größe `VMEM_PAGESIZE` unterteilt. Diese werden in der Datei `pagefile` hintereinander abgelegt.
  Somit nimmt der Prozess `mmanage` eine Seite aus der Datei `pagefile` und lagert sie in einen Seitenrahmen von `mainMemory` ein. Wenn eine Seite aus einem Seitenrahmen verdrängt wird, dann schreibt `mmanage` die modifizierte Seite an die richtige Stelle in der Datei `pagefile` zurück.

- Weiterhin liegt die Seitentabelle (page table) im Shared Memory mit dem `SHMKEY ./src/vmem.h` . Die Seitentabelle wird vom Prozess `mmanage` gepflegt und von dem Modul `vmaccess` , das im Prozess `vmappl` eingebunden ist, gelesen.
- Der Prozess `vmappl` stellt eine Anwendung dar. Sie füllt zuerst ein Feld mit Zufallszahlen und sortiert es anschließend. Das Feld liegt im virtuellen Speicher. Somit lädt `mmanage` die jeweils benötigten Seiten in einen Seitenrahmen.
  Der zweite Teil steht im Modul `vmaccess.c` und bildet über die beiden Funktionen `vmem_read` und `vmem_write` die Schnittstelle zum Speicher.
- Das zweite Shared memory wird im Modul `syncdataexchange.c` implementiert. Über zwei Semaphore, die im diesem Speicher liegen, wird der Zugriff auf die Seitentabelle und den Hauptspeicher synchronisiert.
  Weiterhin müssen `vmem_read` und `vmem_write` , die die Umrechnung von virtuellen - in physikalische Adressen vornehmen, den `mmanage` über einen PageFault informieren. Ist dies der Fall, blockt `vmem_read` bzw. `vmem_write` bis `mmanage` mitteilt, dass die zum PageFault gehörige Seite in einem Seitenrahmen eingelagert ist und die Seitentabelle aktualisiert wurde.

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 unter EMIL. 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 `pagefile` 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 EMIL 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` liegt vor.
  Zu Ihrer Unterstützung finden Sie neben dem Code des Anwendungsprogramms einige Hilfsfunktionen und Datenstrukturen im Code unter EMIL. Die entsprechende DoxyGen Dokumentation liegt auch vor.
- Unter EMIL 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 angepassen werden.**
  Ein ähnliches Shell Skript haben Sie in Aufgabe 1 erstellt, so dass Sie das Skript `run_all` leicht verstehen können. Bitte schauen Sie sich das Skript im Detail an, da die dort implementierten Abläufe für das Verständnis der Aufgabe wesentlich sind.
- Ein `Makefile` finden Sie im Code unter EMIL. **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:

In EMIL 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 Log Files generiert.** Es ist wichtig, dass Sie schon vor dem Praktikumstermin sicherstellen, dass Ihre Log Files mit denen aus EMIL 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 Log Files übereinstimmen, dann müssen Sie anhand der Ausgaben Ihres Programms die korrekte Funktionsweise des Aging Algorithmus darstellen können.
- Die Musterlösung und der unter EMIL bereit gestellte Code können fehlerhaft sein. Wenn Sie einen Fehler finden, schicken Sie mit bitte eine E-Mail. Erklären Sie kurz den Fehler.

Hinweise

- Schauen Sie sich die mitgelieferte Software in Ruhe an.
- Setzen Sie Ihre Lösung schrittweise um. Überprüfen Sie nach jedem Schritt, dass die Lösung das gewünschte Verhalten realisiert.
- **Sie müssen auf der mitgelieferten Software aus diesem Semester aufbauen**. Die Grundfunktion, dass Anwendung und Memory Manager separate Programme sind, muss erhalten bleiben. Dies gilt ebenfalls für die synchrone Kommunikation und das Shared Memory für `vmem.h` . Den Rest des Codes können Sie nach belieben ändern, solange die Ergebnisse mit der Musterlösung (Vergleich der LogFiles) übereinstimmen.
- Zu Anfang werden wahrscheinlich die LogFiles des Aging Algorithmus von der Musterlösung abweichen. Ist dies der Fall, überprüfen Sie bitte folgende Punkte Ihrer Implementierung:
  - Wird Aging zum richtigen Zeitpunkt durchgeführt?
  - Führen Sie das Aging durch, nachdem der Page Fault behoben und das Reference Flag gesetzt wurde.
  - Können mehrere Pages aufgrund Ihres Alters ausgelagert werden (haben alle das selbe Alter), lagern Sie die Page mit der höchsten Frame-Nummer aus.
  - Wird eine Seite eingelagert, setzen Sie den Age Zähler auf 0x80. Das ist wichtig, damit die frisch eingelagerte Seite nicht sofort wieder ausgelagert wird,

wenn zwei Page Faults hintereinander auftreten, ohne das zwischenzeitlich das Age neu berechnet wurde.

- Als Anregung finden Sie im mitgelieferten Quellcode und in der Dokumentation die Signaturen der Funktionen der Musterlösung. Sie können natürlich eigene Konzepte verwirklichen.
- Das Pagefile wird mit Zufallszahlen initialisiert, damit Sie im Dump erkennen, ob sich überhaupt was getan hat. Für eine reproduzierbare Initialisierung wurde die Konstante `SEED_PF` eingeführt. Über den Funktionsaufruf `srand(SEED_PF)` wird der Zufallsgenerator initialisiert.
- Alle Signale verwenden den selben Signalhandler. Der Quellcode zur Initialisierung und Installation der Signalhandler wird mitgeliefert.
- Sie benötigen sicherlich den C-Debugger der Eclipse Umgebung. Damit die in der Aufgabe verwendeten Signale nicht mit der Steuerung des Debuggers interferieren, muss der Debugger gemäß der Beschreibung der Datei *README_DEBUGGER* eingestellt werden.