Paging-Simulator:

Eine Umgebung zur Analyse von Seitenersetzungsalgorithmen

Jungmin Son

Oliver Jockel

Agenda

- Motivation & Ziele
- Architekturübersicht
- Implementierung: C/C++
- GUI (Qt)
- TraceLoader & Eingabedateien
- Live-Demonstration: Paging Simulator UI
- Evaluation & Ergebnisse
- Dokumentation
- Praktische Erkenntnisse & Herausforderungen
- Fazit & Ausblick
- Fragen

Motivation & Ziele des Projekts

- Paging ist ein zentrales Konzept moderner Betriebssysteme
- Verständnis von Speicherverwaltung erfordert praktische Experimente
- Viele Algorithmen (FIFO, LRU, NFU, NRU …) lassen sich theoretisch erklären, aber schwer „sichtbar“ machen
- Eine Simulationsumgebung mit GUI macht das Verhalten von Paging **anschaulich und nachvollziehbar**

Ziele

- Entwicklung einer **flexiblen Simulationsumgebung** für Paging-Algorithmen
- Bereitstellung einer **API**, um neue Algorithmen einfach einzubinden
- Visualisierung von:
  - Physischem Speicher, Seitentabelle, TLB
  - Page Faults, TLB Hits/Misses, Statistiken
- Evaluation klassischer Algorithmen (FIFO, Second Chance, LRU, NRU, NFU mit/ohne Aging)
- Erweiterbarkeit für zukünftige Experimente und Lehre

Architekturübersicht

Gesamtstruktur des Paging-Simulators

Simulation (Core)

- `Simulation.cpp` / `Simulation.h`
- EventQueue für Ablaufsteuerung
- Verwaltung von Prozessen, Seiten, Frames

TraceLoader

- Liest `trace.txt` → erzeugt MemoryAccessEvents
- Unterstützt **Read (R)** und **Write (W)**

Algorithmen (Strategien)

- Interface: `PagingAlgorithm`
- Implementierungen: FIFO, Second Chance, LRU, NRU, NFU (mit/ohne (mit/ohne Aging)

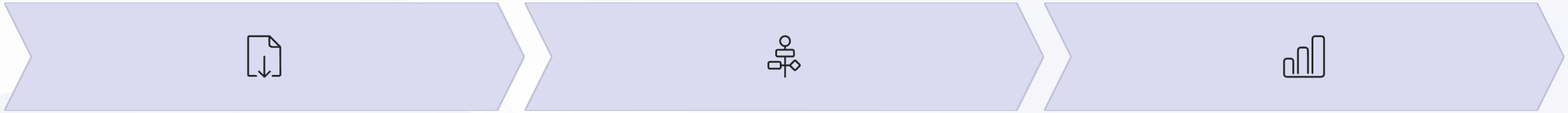
GUI (Qt)

- `ConfigurationWindow`: Parameter-Eingabe
- `SimulationWindow`: Visualisierung (RAM, Seitentabelle, TLB, Log, TLB, Log, Trace Preview)

Architekturübersicht

Datenfluss

Ablauf des Datenflusses



1. TraceLoader lädt Eingabe

Verarbeitet `trace.txt` und generiert MemoryAccessEvents.

2. EventQueue verarbeitet

Steuert den Simulationsablauf und ruft den `PagingAlgorithm` auf.

3. Ergebnisse an die GUI

Page Faults, TLB-Hits und Speicherzustände werden visualisiert.

Implementierung: C/C++ Kernmodule

Algorithmen-API

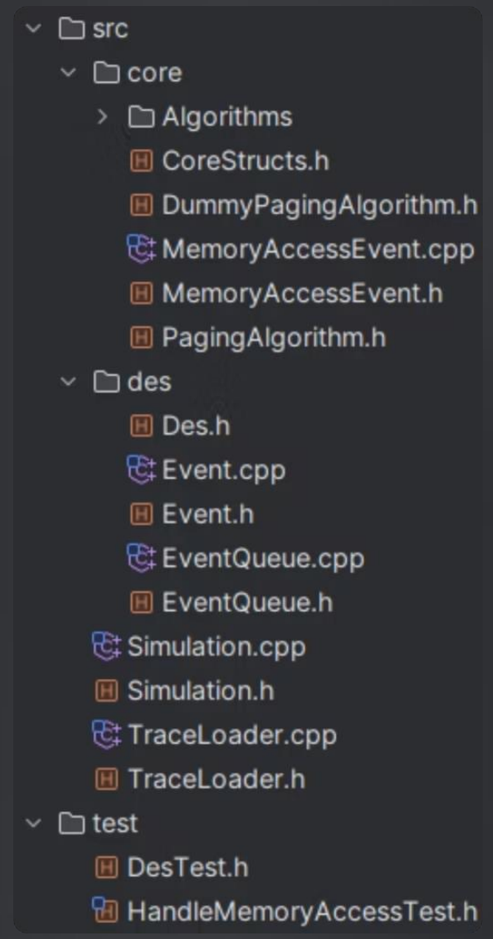
- Gemeinsames Interface:

```
virtual void memoryAccess(int pageId) = 0;
virtual int  selectVictimPage() = 0;
virtual void pageLoaded(int pageId, int frameIndex) = 0;
```

- API erzwingt saubere Trennung
  - Algorithmus verwaltet interne Daten (z.B. Zähler, Zeitstempel)
  - Simulator aktualisiert Speicher & Seitentabellen
- Implementierungen: FIFO, Second Chance, LRU, NRU, NFU (mit/ohne Aging)
- Beispiel: FIFO

```
int FIFOAlgorithm::selectVictimPage() {
    if (queue.empty()) throw std::logic_error("FIFO: no page to evict");
    int victim = queue.front();
    queue.pop();
    return victim;
}

void FIFOAlgorithm::pageLoaded(int pageId, int frameIndex) {
    queue.push(frameIndex);
}
```



Implementierung: C/C++ Kernmodule

Simulation

Die Simulation verwaltet Hauptspeicher, Prozesse und bietet Schnittstellen zu den Paging-Algorithmen.

handleMemoryAccess

- **Koordinator** für jeden Speicherzugriff
- **Lookup:** TLB → Seitentabelle
- **Ergebnisse:** TLB-Hit, Page-Hit, Page-Fault
- **Bei Treffer:** Status-Bits (R/D) & TLB aktualisieren
- **Bei Fehler:** Delegiert an `handlePageFault`

handlePageFault

- **Trigger:** Nur bei Page-Fault
- **Ziel:** Freien Rahmen finden
- **Speicher voll?** → Algorithmus wählt Opfer
- **Opfer:** Aus Seitentabelle & TLB entfernen
- **Neue Seite:** In Rahmen laden, Seitentabelle & TLB aktualisieren

Die Module umfassen zudem die Statistik-Erfassung von Zugriffen, Page Faults und TLB Hits/Misses.

TraceLoader & Eingabedateien

Eingabedatei (trace.txt)

- **Format:**

```
pageId [R|W]
0 R
0 R
1 W
2 R
3 W
2 R
0 R
```

- **pageId**: Virtuelle Seitennummer
- **R/W**: Art des Zugriffs
  - `R` = Read → Lesezugriff
  - `W` = Write → Schreibzugriff (Seite als „dirty“ markiert)

TraceLoader

- Liest Datei zeilenweise
- Erzeugt für jede Zeile ein `MemoryAccessEvent`
- Übergibt Events an die **EventQueue (DES)**

- Unterstützt **flexible Szenarien**:
  - Sequenzielle Zugriffe
  - Schreibintensive Workloads
  - Realitätsnahe Simulation

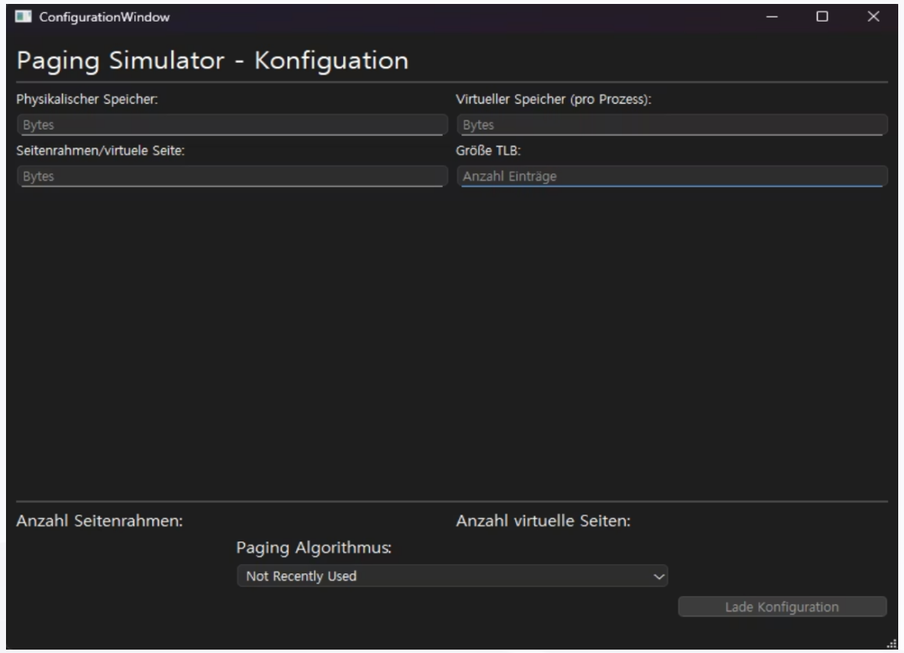
Motivation für R/W

- Nur Seiten-IDs reichen nicht → keine Unterscheidung zwischen Lesen und Schreiben
- Schreibzugriffe sind teurer (Dirty-Bit, ggf. ggf. Rückschreiben in den Speicher)

- Algorithmen wie NRU sind auf Dirty-Bits angewiesen

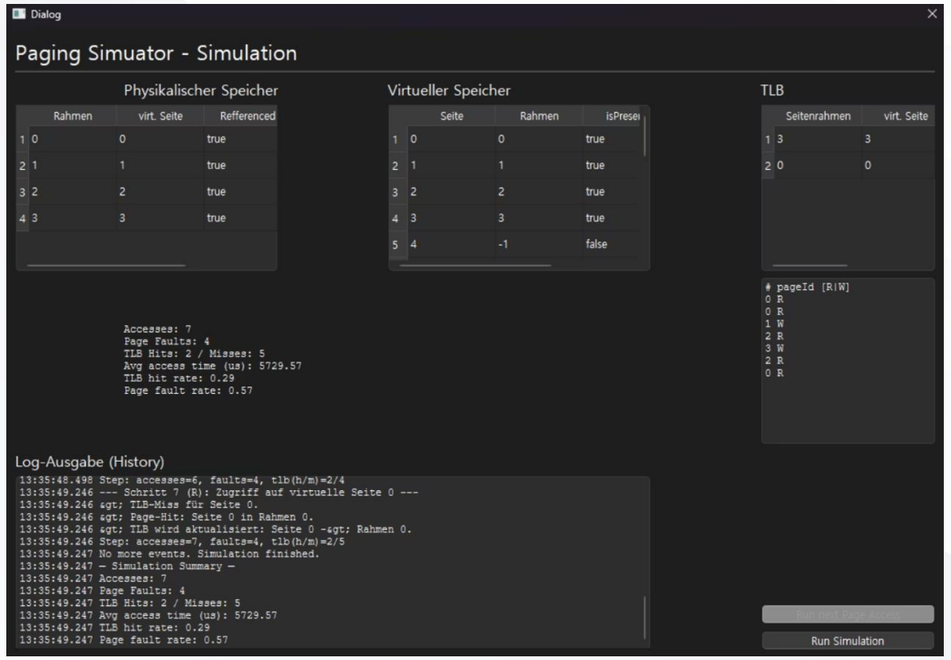
Intuitive GUI mit Qt

Konfigurationsfenster:



- Eingabe der Simulationsparameter:
  - Physischer Speicher
  - Virtueller Speicher
  - Seitengröße
  - TLB-Größe
  - Auswahl des Algorithmus

Simulationsfenster:



- Tabellen:
  - Physischer Speicher
  - Seitentabelle (Virtueller Speicher)
  - TLB
- Log-Ausgabe mit Zeitstempeln
- Trace-Vorschau (`trace.txt`)

Live-Demonstration: Paging Simulator UI

—Simulation mit FIFO

Evaluation & Ergebnisse

Vergleich der Algorithmen

- **FIFO**: Einfach, aber kann Anomalien zeigen
- **Second Chance**: Verbesserung von FIFO durch Referenced Bit
- **LRU**: Gute Approximation der „optimalen“ Strategie, aber teuer in Realität
- **NRU**: Klassifiziert Seiten (R/D Bits) → zufällige Auswahl aus Klassen
- **NFU (no aging)**: Zählt Gesamtzugriffe → „Elefanten-Gedächtnis“
- **NFU (mit aging)**: Berücksichtigt Zeit → bessere Anpassung an Phasenwechsel

Beobachtungen

- TLB Hits reduzieren signifikant die mittlere Zugriffszeit
- Page Fault Rate variiert je nach Algorithmus und Trace
- Aging-Mechanismen passen sich dynamisch besser an

Evaluation & Ergebnisse

Visualisierung

- Statistiken: Zugriffe, Page Faults, TLB Hits/Misses, Hit-Rate
- Log-Ausgabe mit Zeitstempeln → transparente Nachvollziehbarkeit

```
Accesses: 7
Page Faults: 4
TLB Hits: 2 / Misses: 5
Avg access time (us): 5729.57
TLB hit rate: 0.29
Page fault rate: 0.57
```

Log-Ausgabe (History)

```
13:35:48.498 Step: accesses=6, faults=4, tlb(h/m)=2/4
13:35:49.246 --- Schritt 7 (R): Zugriff auf virtuelle Seite 0 ---
13:35:49.246 &gt; TLB-Miss für Seite 0.
13:35:49.246 &gt; Page-Hit: Seite 0 in Rahmen 0.
13:35:49.246 &gt; TLB wird aktualisiert: Seite 0 -&gt; Rahmen 0.
13:35:49.246 Step: accesses=7, faults=4, tlb(h/m)=2/5
13:35:49.247 No more events. Simulation finished.
13:35:49.247 — Simulation Summary —
13:35:49.247 Accesses: 7
13:35:49.247 Page Faults: 4
13:35:49.247 TLB Hits: 2 / Misses: 5
13:35:49.247 Avg access time (us): 5729.57
13:35:49.247 TLB hit rate: 0.29
13:35:49.247 Page fault rate: 0.57
```

Dokumentation

1

Code-Kommentare (Doxygen)

- Alle Klassen, Methoden und Attribute dokumentiert
- Automatische Generierung von HTML/PDF-API-Referenzen möglich

2

README-Dateien

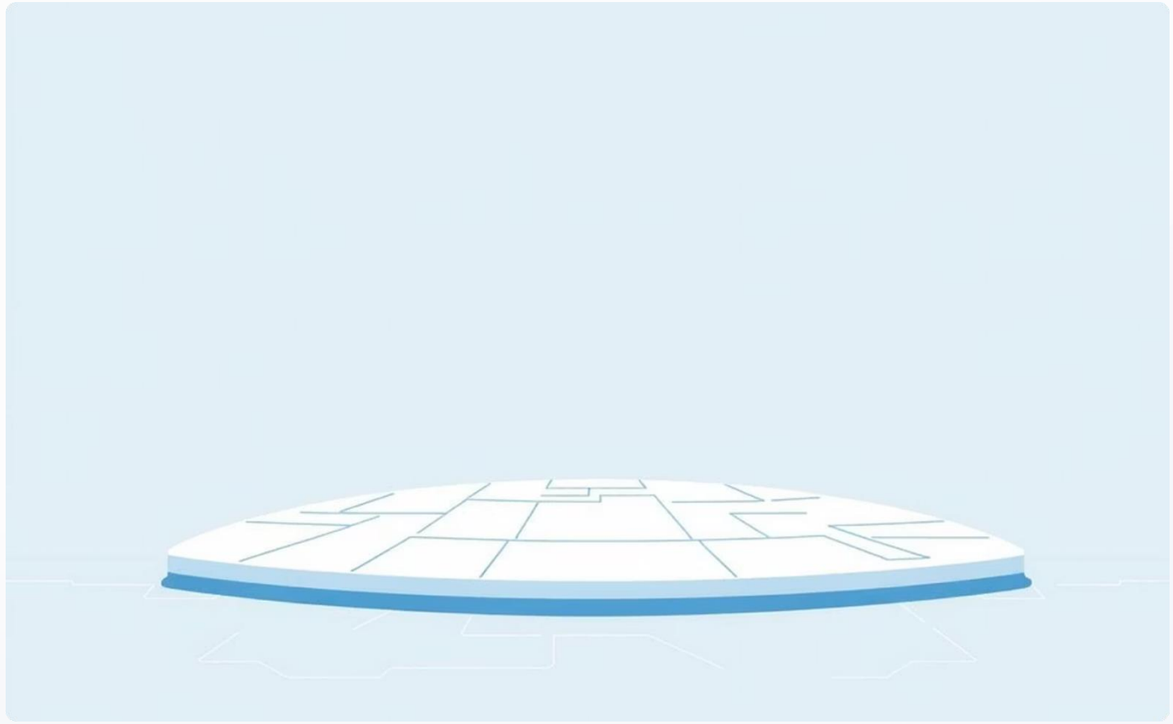
Separate README-Dateien für den `PagingSimulator` und die `PagingSimulatorUI` bieten detaillierte Anleitungen zur Kompilierung, Installation und Nutzung auf verschiedenen Betriebssystemen (Windows, Linux, macOS).

3

Projekt-Dokumentation

Eine umfassende Projektdokumentation beinhaltet eine detaillierte Projektbeschreibung, Implementierungsdetails, API-Referenz Implementierungsdetails, API-Referenz (UML-Diagramme), sowie eine Strukturübersicht. Dies gewährleistet die Nachvollziehbarkeit Nachvollziehbarkeit und Wartbarkeit des Projekts.

Praktische Erkenntnisse & Mögliche Erweiterungen



Erkenntnisse

- Diskrete-Ereignis-Simulation (DES) bietet eine flexible Grundlage für Paging-Paging-Algorithmen
- Unterschiedliche Algorithmen zeigen stark divergierende Page-Fault-Raten je Raten je nach Workload
- GUI-Integration erleichtert das Verständnis der Abläufe (didaktischer Mehrwert)
- Doxygen-Kommentierung zwingt zu klarer Schnittstellendefinition

Mögliche Erweiterungen

- Implementierung weiterer Algorithmen: z. B. Clock, Optimal (Belady), Working-Set
- Visualisierung: Heatmaps der Seitennutzung, Zeitachsen für Page Faults
- Unterstützung größerer Trace-Formate (aus realen Workloads)
- Parametrisierbare Zeitbasis im DES (z. B. Aging alle N Zyklen)
- Export der Simulationsergebnisse (CSV, JSON) für externe Analyse

Fazit & Ausblick

Fazit

- Implementierung einer **kompakten Paging-Simulationsumgebung (Prototyp)**
- Unterstützung mehrerer klassischer Algorithmen (FIFO, Second Chance, LRU, NRU, NFU ± Aging)
- **GUI (Qt)** ermöglicht intuitive Visualisierung → hoher didaktischer Nutzen
- **DES-basierter Ansatz**: modular, erweiterbar, realitätsnah im *kleinen Maßstab*
- Dokumentation (Doxygen, Benutzerhandbuch) sorgt für Nachvollziehbarkeit und Wartbarkeit

Ausblick

- Erweiterung um zusätzliche Algorithmen (Clock, Optimal, Working-Set)
- Integration realitätsnäherer Workloads und größerer Traces
- Erweiterte Visualisierung (Heatmaps, Zeitachsen, Statistiken)
- Export/Analyse der Ergebnisse in externen Tools (CSV, JSON, Python)
- Nutzung als **Lehr- und Lernplattform** für Betriebssystem-Kurse

Fragen