Dokumentation der Implementierungsschritte – OSMP

Zielsetzung

Entwicklung einer funktionalen, robusten und skalierbaren Bibliothek für die Interprozesskommunikation auf Basis von Shared Memory und POSIX-Synchronisationsmechanismen. Der Fokus liegt auf einer strukturierten Nachrichtenübermittlung zwischen Prozessen mit Synchronisation über Semaphore und geschützten Datenstrukturen.

Struktur der OSMP-Implementierung

1. Zentrale Struktur: osmp_shared_info_t

- Dynamisch allokiert im Shared Memory
- Beinhaltet:
    - Prozessanzahl
    - pid_map[] für Rangermittlung
    - Pointer auf Mailboxen
    - Slot-Verwaltung über FreeSlotQueue
    - MessageSlot[] für Nachrichten
    - Logging-Konfiguration (Pfad, Level)

2. Initialisierung durch ompRun

- Liest Prozessanzahl und Executable
- Erstellt Shared Memory via shm_open, ftruncate, mmap
- Initialisiert:
    - Mailboxen (sem_space, sem_data, mutex)
    - Slot-Queue (free_slots[], head, tail, mutex, sem_slots)
    - Loggingstruktur (Pfad wird via Argument übergeben, siehe TODO)
- Forkt Prozesse + ersetzt sie via execvp()

3. OSMP_Init()

- Jeder Kindprozess ruft OSMP_Init() auf
- Bestimmt eigene pid → sucht zugehörigen rank
- Initialisiert Zugriff auf Shared Memory per mmap

- Stellt Verbindungen zu eigener Mailbox, Slotqueue etc. her

Implementierung der Kommunikationsfunktionen

OSMP_Send() Ablauf:

1. sem_wait(sem_slots) → Slotverfügbarkeit sicherstellen
2. mutex-Schutz für Zugriff auf FreeSlotQueue
3. Slot-ID aus Queue lesen
4. Nachricht in Slot schreiben (inkl. Absender, Typ, Payload)
5. Slot in Mailbox des Empfängers einfügen
    - ggf. FIFO-Verkettung über next
6. sem_post(sem_data) für Empfänger

OSMP_Recv() Ablauf:

1. sem_wait(sem_data) → auf neue Nachricht warten
2. mutex-geschützter Zugriff auf head der Mailbox
3. Slot-ID entnehmen, Nachricht lesen
4. Slot zurück in FreeSlotQueue über tail
5. sem_post(sem_slots)

Speicherlayout und Synchronisation

- Jede Mailbox ist ein ringgepufferter Nachrichtenspeicher mit Semaphoren:
    - sem_space: freier Platz
    - sem_data: empfangsbereite Nachrichten
    - mutex: Schutz für head/tail
- Die zentrale SlotQueue arbeitet ebenfalls ringgepuffert, gesteuert über:
    - head, tail, free_slots[]
    - sem_slots und mutex

Erweiterungspotenzial (noch offen)

- Konfigurierbarer Logfile-Pfad aus argv

- Barrier-Funktion für globale Synchronisation
- Nicht-blockierende Varianten (OSMP_TrySend, OSMP_TryRecv)
- Testframework mit 3+ Prozessen zur systematischen Analyse von Race Conditions

---

Fazit

Die Kernfunktionen von OSMP sind implementiert und synchronisiert. Die Struktur erlaubt die problemlose Kommunikation zwischen beliebig vielen Prozessen bei voller Kontrolle über Puffer, Synchronisation und Deadlockvermeidung. Die Implementierung folgt den Lernzielen des Praktikums 3 und bietet eine solide Grundlage für Erweiterungen.