OSMP Runner – Technische Dokumentation

Projekt: Betriebssysteme Praktikum – OSMP

Autor: Konrad Skwarski, Christopher Jung, Erik Kolke

Datum: 2025-04-02

Dateien: ospmRun.c, ospmRun.h

Ziel: Initialisierung und Ausführung einer verteilten, message-passing-basierten IPC-Laufzeitumgebung unter UNIX

1. Zweck des Programms

Der ospmRunner ist ein Initialisierungs- und Steuerprogramm, das eine vordefinierte Anzahl an OSMP-Prozessen erzeugt. Diese Prozesse kommunizieren über gemeinsamen Speicher (shared memory) mit Message Passing. Der Runner übernimmt:
- das Setup des Shared Memorys,
- die Initialisierung von Semaphoren, Mailboxen, Synchronisationsstrukturen,
- das Starten der Prozesse via fork() und execvp(),
- das Logging aller relevanten Aktionen und Fehlermeldungen,
- das koordinierte Warten auf Prozessende (waitpid()).

2. Kompilierung

gcc -o ospmRunner ospmRun.c ../ospmLibrary/ospmLib.c -lpthread -lrt

3. Ausführungsparameter

./ospmRunner -p <prozessanzahl> -l <logfile> -v <verbosity> -e <pfad_zur_exe> [args...]

4. Globale Variablen

- logfile_path: Pfad zur Logdatei
- verbosity_level: Verbositätslevel (1: Standard, 2: erweitert, 3: Debug)
- buffer[256]: Buffer zur temporären Formatierung von Logausgaben
- osmp_shared: Zeiger auf den gemeinsam genutzten Speicherbereich

5. Funktionsübersicht

5.1 setup_shared_memory(int process_count)

Zweck:
Initialisiert den Shared Memory, alle Mailboxen, Slot-Queues, Semaphore und die Barrierestruktur für process_count Prozesse.

Ablauf:
- Speichergröße berechnen
- Shared Memory anlegen (shm_open + ftruncate)
- Mapping mit mmap
- Initialisierung:
  - pid_map: Zuweisung Rank → PID
  - Mailboxen: Semaphore, Mutex, Slot-Index-Array
  - FreeSlotQueue
  - Logging-Mutex
  - Barriere

Rückgabe:
- 0 bei Erfolg
- EXIT_FAILURE bzw. OSMP_FAILURE bei Fehler

5.2 main(int argc, char *argv[])

Zweck:
Hauptfunktion zum Starten des OSMP-Systems. Parst Argumente, richtet das Umfeld ein und startet Kindprozesse mit execvp().

Ablauf:
1. Argumente parsen via getopt
2. setup_shared_memory aufrufen
3. Logfile vorbereiten
4. Prozesse mit fork() starten
5. execvp() im Kindprozess
6. PID-Map im Elternprozess
7. Logging & waitpid() auf alle Kinder
8. Aufräumen

Rückgabe:
- EXIT_SUCCESS bei Erfolg
- EXIT_FAILURE bei Fehler

6. Initialisierte Strukturen (aus osmpLib.h)

6.1 MailboxTypeManagement

- sem_free_mailbox_slots: Steuerung freier Nachrichtenslots
- sem_msg_available: Steuerung empfangsbereiter Nachrichten

- mailbox_mutex: Schutz für kritische Abschnitte
- slot_indices[OSMP_MAX_SLOTS]: Index-Mapping für Nachrichten
- in, out: Zeiger für zirkuläre Mailboxen

6.2 FreeSlotQueue

- sem_slots: Zähler für verfügbare Nachrichtenslots
- free_slots_mutex: Schutz der Queue
- free_slots[OSMP_MAX_SLOTS]: Slot-ID-Puffer
- head, tail: FIFO-Verwaltung

6.3 osmp_shared_info_t

- process_count
- logfile_path[256]
- verbosity_level
- log_mutex
- pid_map[]
- mailboxes[]
- fsq (FreeSlotQueue)
- barrier (Synchronisation)

7. Logging & Fehlermanagement

Alle Vorgänge wie Prozessstarts, fork-/exec-Fehler und Exit-Codes werden mit OSMP_Log() protokolliert.
Das Logging erfolgt synchronisiert über sem_wait(&log_mutex).

8. Beispielaufruf

./ospmRunner -p 4 -l ./osmp.log -v 2 -e ./echoAll A B C

→ Startet 4 Prozesse mit dem Programm ./echoAll,
schreibt Logeinträge in osmp.log und setzt das Logging-Level auf 2.

9. Weiterführende Literatur & Referenzen

- POSIX man pages: man 2 fork, man 3 sem_init, man 2 mmap, man 3 pthread_mutex_init

- Vorlesungsfolien Betriebssysteme – FH Münster (Prof. Dr.-Ing. Malysiak)
- OSMP Praktikumsbeschreibung: BS-Praktikumsbeschreibung2025.pdf

10. Funktionsparameter & Rückgabewerte

setup_shared_memory(int process_count)

Initialisiert Shared Memory, alle Mailboxen und Synchronisationsmechanismen.

- Parameter:
    - int process_count – Anzahl der zu initialisierenden Prozesse

Rückgabewert: int – 0 bei Erfolg, EXIT_FAILURE oder OSMP_FAILURE bei Fehler

main(int argc, char *argv[])

Startpunkt des Programms. Parst Kommandozeilenargumente, initialisiert Umgebung und startet Prozesse.

- Parameter:
    - int argc – Anzahl der Kommandozeilenargumente
    - char *argv[] – Kommandozeilenargumente (z. B. -p, -l, -v, -e)
    - Rückgabewert: int – EXIT_SUCCESS bei Erfolg, EXIT_FAILURE bei Fehler

OSMP Library – Technische Dokumentation

Projekt: Betriebssysteme Praktikum – OSMP

Autor: Konrad Skwarski, Christopher Jung, Erik Kolke

Dateien: osmpLib.c, osmpLib.h

Stand: 2025-04-02

1. Zweck der Bibliothek

Die OSMP Library implementiert eine message-passing-basierte Interprozesskommunikation über Shared Memory. Sie nutzt Semaphoren und Mutexes zur Synchronisation und unterstützt blockierende Kommunikation, Barrieren sowie kollektive Operationen.

2. Globale Variablen

- osmp_shared: Zeiger auf das zentrale Shared Memory Segment
- osmp_rank: Rang des aktuellen Prozesses
- mailboxes: Mailboxverwaltung für Prozesse
- fsq: Verwaltung freier Nachrichtenslots
- slots: Nachrichtenspeicher (Slots)

3. Initialisierung

Die Initialisierung erfolgt über OSMP_Init() und richtet alle Strukturen ein, ermittelt den Prozess-Rang und stellt den Zugriff auf geteilten Speicher her.

4. Funktionen

OSMP_Init(const int *argc, char ***argv)

Initialisiert die OSMP-Umgebung und ermittelt den Rank.

- Parameter:
    - argc: Kommandozeilenparameteranzahl (nicht verwendet)
    - argv: Kommandozeilenargumente (nicht verwendet)

Rückgabewert: OSMP_SUCCESS bei Erfolg, OSMP_FAILURE bei Fehler

OSMP_Finalize(void)

Beendet den Prozess und gibt Ressourcen frei.

Rückgabewert: OSMP_SUCCESS bei Erfolg, OSMP_FAILURE bei Fehler

OSMP_Send(const void *buf, int count, OSMP_Datatype datatype, int dest)

Sendet eine Nachricht an einen anderen Prozess.

- Parameter:
    - buf: Zeiger auf Sendepuffer
    - count: Anzahl der Elemente
    - datatype: Datentyp der Elemente
    - dest: Zielprozess (Rang)

Rückgabewert: OSMP_SUCCESS bei Erfolg, OSMP_FAILURE bei Fehler

OSMP_Recv(void *buf, int count, OSMP_Datatype datatype, int *source, int *len)

Empfängt eine Nachricht aus der eigenen Mailbox.

- Parameter:
  - buf: Zeiger auf Empfangspuffer
  - count: maximale Anzahl
  - datatype: erwarteter Datentyp
  - source: Adresse für Senderang
  - len: Adresse für empfangene Bytes

Rückgabewert: OSMP_SUCCESS bei Erfolg, OSMP_FAILURE bei Fehler

OSMP_Size(int *size)

Gibt die Anzahl gestarteter Prozesse zurück.

- Parameter:
  - size: Rückgabewert

Rückgabewert: OSMP_SUCCESS bei Erfolg, OSMP_FAILURE bei Fehler

OSMP_Rank(int *rank)

Gibt den Rang des aktuellen Prozesses zurück.

- Parameter:
  - rank: Rückgabewert

Rückgabewert: OSMP_SUCCESS bei Erfolg, OSMP_FAILURE bei Fehler

OSMP_SizeOf(OSMP_Datatype datatype, unsigned int *size)

Liefert die Größe eines OSMP-Datentyps.

- Parameter:
  - datatype: Typ der Daten
  - size: Rückgabewert für Byte-Größe

Rückgabewert: OSMP_SUCCESS bei Erfolg, OSMP_FAILURE bei Fehler

OSMP_Barrier(void)

Synchronisiert alle Prozesse an einer Barriere.

Rückgabewert: OSMP_SUCCESS bei Erfolg, OSMP_FAILURE bei Fehler

OSMP_Gather(void *sendbuf, int sendcount, OSMP_Datatype sendtype, void *recvbuf, int recvcount, OSMP_Datatype recvtype, int root)

Sammelt Daten von allen Prozessen beim Root.

- Parameter:
    - sendbuf: lokaler Sendepuffer
    - sendcount: Anzahl zu sendender Elemente
    - sendtype: Datentyp
    - recvbuf: Empfangspuffer (nur Root)
    - recvcount: Anzahl je Prozess
    - recvtype: Empfangs-Datentyp
    - root: Rank des Root-Prozesses

Rückgabewert: OSMP_SUCCESS bei Erfolg, OSMP_FAILURE bei Fehler

OSMP_Log(OSMP_Verbosity verbosity, char *message)
Schreibt eine Nachricht in die Logdatei.

- Parameter:
    - verbosity: Log-Level
    - message: Nachrichtentext

Rückgabewert: OSMP_SUCCESS bei Erfolg, OSMP_FAILURE bei Fehler

5. Datenstrukturen

MailboxTypeManagement
- Steuerung des Nachrichtenpuffers eines Prozesses mit Semaphoren und Mutex

MessageType
- Struktur einer Nachricht mit Typ, Quelle, Länge und Payload

FreeSlotQueue
- Globale Queue zur Verwaltung freier Nachrichtenslots

osmp_shared_info_t
- Struktur für globalen Shared-Memory-Status, inkl. Logging, PID-Map und Barriere

barrier.c / barrier.h – Technische Dokumentation
Projekt: Betriebssysteme Praktikum – OSMP

Autor: Konrad Skwarski, Christopher Jung, Erik Kolke

Datum: 19.05.2025

Dateien: barrier.c, barrier.h

Ziel: Prozessübergreifende Barriere für Synchronisation mittels POSIX-Mutex und -Condition

1. Zweck der Barriere

Die Datei `barrier.c/h` implementiert eine prozessübergreifende Barriere zur Synchronisation. Eine definierte Anzahl an Prozessen/Threads wartet an der Barriere, bis alle anderen sie erreicht haben. Dann werden alle gleichzeitig freigegeben.

2. Datenstruktur: barrier_t

- pthread_mutex_t mutex – schützt den Zugriff auf interne Variablen
- pthread_cond_t convar – wartet auf Erreichen der Bedingung
- int valid – Kennzeichnung, ob Barriere initialisiert wurde
- int threshold – Anzahl erwarteter Threads
- int counter – verbleibende Threads
- int cycle – Barrierenummer zur Wiedererkennung neuer Zyklen

3. Funktionen & Funktionsweise

barrier_init(barrier_t *barrier, int count)

Initialisiert eine Barriere im Shared Memory mit einem erwarteten Teilnehmer-Zähler.

- Parameter:
  - barrier – Zeiger auf die zu initialisierende Barriere
  - count – Anzahl der erwarteten Teilnehmer (muss > 0 sein)

Rückgabewert: 0 bei Erfolg, sonst Fehlercode (z. B. EINVAL)

Funktionsweise:
- Initialisiert prozessübergreifende Mutex- und Condition-Attribute
- Setzt Zähler, Zustand und Zyklusvariablen
- Verwendet: pthread_mutexattr_setpshared, pthread_condattr_setpshared, pthread_mutex_init, pthread_cond_init

barrier_destroy(barrier_t *barrier)

Zerstört die Synchronisationsmechanismen der Barriere.

- Parameter:
  - barrier – Zeiger auf die Barriere

Rückgabewert: 0 bei Erfolg, sonst Fehlercode

Funktionsweise:
- Prüft, ob Barriere gültig ist
- Zerstört Mutex und Condition Variable
- Setzt Barriere als ungültig (valid = 0)
- Verwendet: pthread_mutex_destroy, pthread_cond_destroy

barrier_wait(barrier_t *barrier)

Synchronisiert alle Teilnehmerprozesse an der Barriere.

- Parameter:
  - barrier – Zeiger auf die aktive Barriere

Rückgabewert: 0 bei Erfolg, sonst Fehlercode

Funktionsweise:
- Sperrt Mutex und dekrementiert counter
- Letzter Teilnehmer: setzt neuen Zyklus und broadcasted
- Andere warten per pthread_cond_wait
- Cancel-Status wird vorübergehend deaktiviert (pthread_setcancelstate)
- Verwendet: pthread_cond_broadcast, pthread_cond_wait, pthread_mutex_unlock