FH MÜNSTER
University of Applied Sciences

Fachbereich
Elektrotechnik und Informatik

Dokumentation und Bericht zur OSMP-Bibliothek

Praktikum Betriebssysteme,
SoSe 2024, PG 1, Gruppe 3

13. Juni 2024
Thomas Fidorin, Alina Rölver
fidorin.thomas@fh-muenster.de
alina.roelver@fh-muenster.de

Inhaltsverzeichnis

1 Einleitung

In diesem Dokument erläutern wir die Funktionsweise unserer Implementierung der OSMP-Bibliothek im Rahmen des Betriebssysteme-Praktikums im Sommersemester 2024. Dieser Bericht erläutert die folgenden drei Hauptkomponenten der Bibliothek:

- den Runner, implementiert in `osmp_runner/osmp_run.c`
- die eigentliche Implementierung der OSMP-Funktionalitäten in `osmp_library/osmplib.c` und `osmp_library/osmplib.h`
- den Logger, implementiert in `osmp_library/logger.c`

Der folgende Bericht fokussiert sich auf die Datenstrukturen und die Synchronisationskonzepte, die unserer Implementierung zugrunde liegen. Die Code-Dokumentation der einzelnen Funktionen findet sich in der Doxygen-Dokumentation im Anhang.

2 Runner

Im Runner werden zunächst die Kommandozeilenargumente geparst und die entsprechenden Einstellungen (Anzahl der Prozesse, Executable, ggfs. Argumente für das Executable, ggfs. Log-Datei und -Verbosität) gesetzt. Auf Basis der Anzahl der Prozesse kann die Größe des Shared Memory berechnet werden. Der Runner öffnet den Shared Memory und initialisiert ihn mit den entsprechenden Anfangswerten und Einstellungen (vgl. dazu Abschnitt 3.1).

Nach der erfolgreichen Initialisierung werden die einzelnen Executable-Prozesse mittels `fork()` gestartet (vgl. die Funktion `start_all_executables()`) und mit `exec()` wird das Executable in den neuen Prozess geladen. Dafür erhält der Runner einen Lock auf das Init-Mutex (vgl. Abschnitt 3.1), den die einzelnen Prozesse in ihrer eigenen Initialisierung (`OSMP_Init()`) ebenfalls locken müssen. Mit einer dazugehörigen Conditon-Variable können sie passiv darauf warten. So wird sichergestellt, dass der Runner erst alle Prozesse und die dazugehören Informationen im Shared Memory vollständig initialisiert hat, bevor ein Prozess darauf zugreifen kann. Falls der `fork()`-Call für einen Prozess fehlschlägt, killt der Runner alle Prozesse beendet das Programm.

Im Erfolgsfall wartet der Runner dann auf das Ende der einzelnen Prozesse, um danach Datenstrukturen im Shared Memory (insbesondere Mutexe und Semaphoren) wieder zu zerstören und den Shared Memory wieder freizugeben.

3 OSMPLib

Die OSMPLib bietet die Funktionalität für die Nutzer unserer Bibliothek, darunter haben wir die Funktionen: OSMP_Send, OSMP_Recv, OSMP_ISend, OSMP_IRecv, OSMP_Barrier und OSMP_Gather sowie die Struktur des Shared Memory.

3.1 Struktur des Shared Memory

Die OSMP-Bibliothek soll dazu dienen, Nachrichten zwischen Prozessen auszutauschen. Für die Implementierung ist zunächst die Unterscheidung zwischen Nachrichtenslot und Postfach wichtig: In einem Nachrichtenslot wird die eigentliche Nachricht gespeichert. Im Postfach eines Prozesses finden sich Verweise auf die Nachrichtenslots, in denen Nachrichten für den jeweiligen Prozess bereit liegen.

An den Anfang des Shared Memory (vgl. Abb. 3.1 auf S. 4)[1] wird die Größe des Memory geschrieben (**Size**, Z. 1). Durch diese Position am Anfang des Speicherbereichs können die Prozesse beim Initialisierungsvorgang, auch schon ohne die Gesamtgröße des Shared Memory zu kennen, diese Information auslesen und so die Gesamtgröße des Shared Memorys berechnen. Weiterhin liegen im Shared Memory ein **Logging-Mutex** für den Zugriff auf die Logdatei (Z. 1) sowie ein **Init-Mutex** und eine **Init-Condition-Variable** für den Initialisierungsvorgang (beides Z. 1; vgl. die Erläuterung dazu in 2).

Weiterhin enthält das Shared Memory eine Liste bzw. Array mit `OSMP_MAX_SLOTS` Elementen (**Free Slots**, Z. 2), in dem die aktuell freien Nachrichtenslots verzeichnet sind. Der **Free-Slots-Index** (Z. 2) indiziert die Stelle in **Free Slots**, an der das nächste freie Postfach verzeichnet ist. (Vgl. dazu auch Abschnitt 3.1.1.) Mit dem Semaphore **sem_shm_free_slots** (Z. 2) wird die Anzahl der freien Nachrichtenslots gezählt. Bei Belegen eines Slots wird die Anzahl mit `sem_wait()` dekrementiert, bei Freigeben eines Slots mit `sem_post()` inkrementiert. Der Mutex **mutex_shm_free_slots** (Z. 2) synchronisiert den Zugriff auf die **Free-Slots**-Liste und ihren Index.

An diese Elemente, die der Verwaltung des Shared Memory und der Nachrichtenslots dienen, schließen sich die `OSMP_MAX_SLOTS` **Message-Slots** an (Z. 3-6). Ein Message-Slot besteht jeweils aus den folgenden Elementen:

- Absender
- Länge

[1] Im weiteren Verlauf dieses Abschnitts referenzieren wir einzelne Stellen der Abb. 3.1 durch Zeilenangaben. Diese beziehen sich auf die farblich markierten Zeilen im Schema. Die Länge bzw. Größe der einzelnen Elemente steht nicht im Zusammenhang mit der tatsächlichen Größe im Speicherbereich. Es handelt sich dabei also nicht um ein *byte alignment*, sondern die Aufteilung in Zeilen dient nur der besseren Darstellbarkeit.

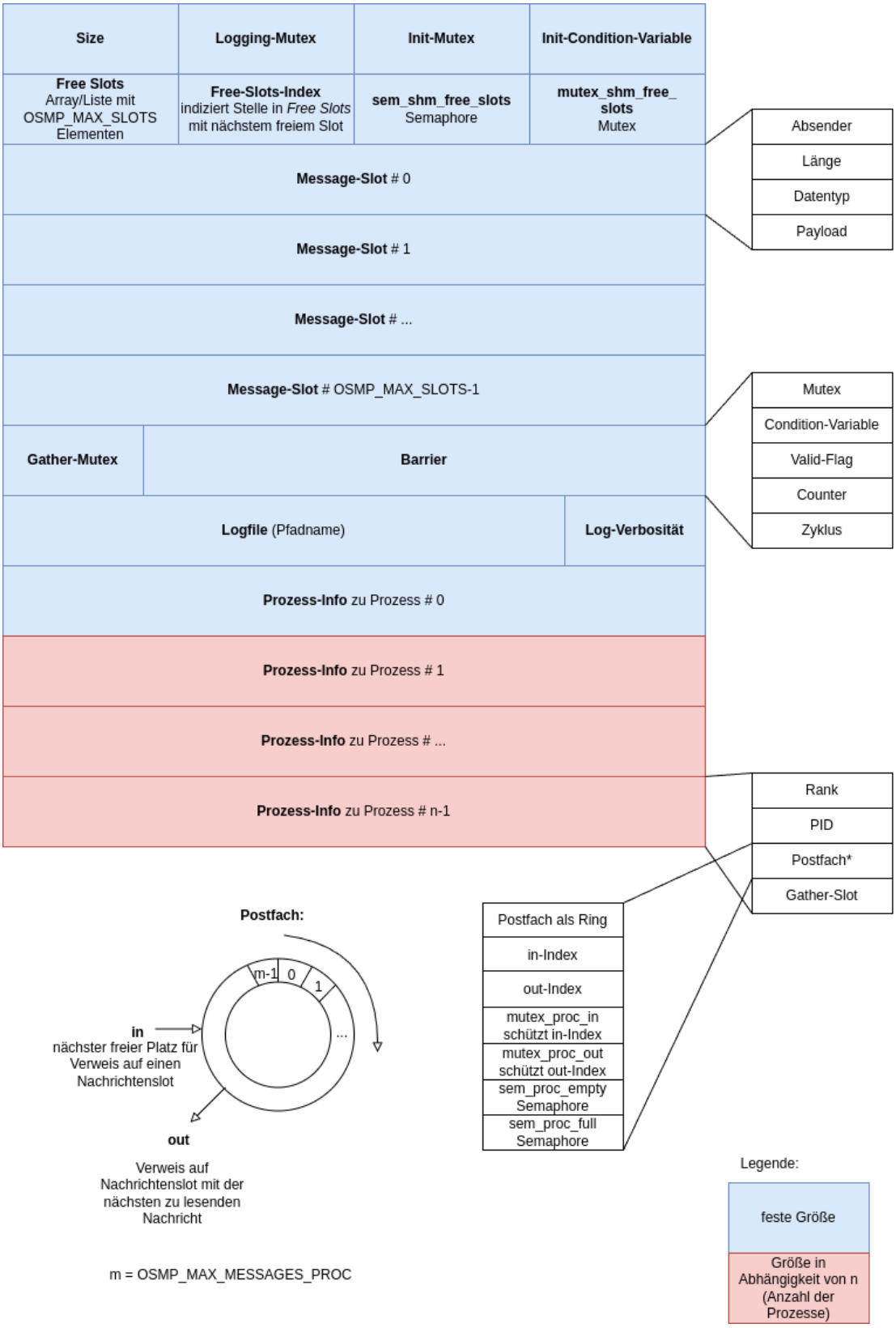


Abbildung 3.1: Schematische Darstellung des Shared Memory und der darin verwendeten Strukturen.

- Datentyp (`OSMP_Datatype`)
- Payload

Mit Hilfe des **Gather-Mutex** (Z. 7) wird der Zugriff auf die Gather-Postfächer synchronisiert (mehr dazu weiter unten). Die **Barrier**-Struktur (Z. 7) dient der Implementierung der Barrier-Funktionalität. Sie besteht aus folgenden Elementen:

- einem Mutex, mit dem der Zugriff auf die weiteren Barrier-Elemente synchronisiert wird
- einer Condition-Variable, mit der passives Warten an der Barriere ermöglicht wird
- einem Valid-Flag, mit dem signalisiert wird, dass die Barriere vollständig initialisiert ist
- einem Counter, mit dem die Anzahl der Prozesse gezählt wird, die die Barriere noch erreichen müssen
- einer Zyklus-Variable, mit der verschiedene aufeinanderfolgende Aufrufe der Barriere unterschieden werden können

Das Shared Memory enthält außerdem den Pfad zum **Logfile** sowie die **Log-Verbosität** (beides Z. 8), da auch auf diese Informationen von allen Prozessen zugegriffen werden muss.

Der Rest des Shared Memory besteht aus **Informationen zu den einzelnen Prozessen** (Z. 9-12). Zu jedem OSMP-Prozess werden die folgenden Informationen verwaltet:

- der Rang des Prozesses
- seine PID
- sein Postfach (vgl. Abschnitt 3.1.2), dies besteht wiederum aus
  - dem eigentlichen Postfach mit `OSMP_MAX_MESSAGES_PROC` Elementen, in denen jeweils ein Verweis auf einen freien Nachrichtenslot stehen kann
  - einem in-Index, der angibt, an welcher Stelle im Postfach ein freier Platz ist, um auf einen Nachrichtenslot zu verweisen
  - einem out-Index, der auf die Stelle im Postfach verweist, die wieder angibt, in welchem Nachrichtenslot sich die nächste zu lesende Nachricht für den Prozess befindet
  - je einem Mutex und einem Semaphore pro in- und out-Index, mit denen die freien Plätze im Postfach synchronisiert (Mutex) bzw. gezählt (Semaphore) werden
- sein Gather-Slot.

Die Anzahl der Prozess-Informationen ist natürlich abhängig von der Anzahl der Prozesse, die als Argument an das Programm übergeben wird. Daher ist auch die gesamte Größe des Shared Memory variabel. Es gibt jedoch einen festen Anteil: Alle Elemente bis einschließlich der ersten Prozess-Info (Z. 1-9) sind fest, da es immer mindestens einen Prozess geben muss. Die festen Elemente sind im Schema blau gekennzeichnet, die variablen rot.

3.1.1 Funktionalität der Free-Slots-Liste

Beim Start des Programms, bzw. genauer nach der Initialisierung des Shared Memory, sind zunächst alle Nachrichtenslots frei. Daher sind in der Free-Slots-Liste zunächst auch alle Slots verzeichnet (vgl. Abb. 3.2). Der Index bewegt sich nach rechts (d.h. er wird inkrementiert), wenn Slots belegt werden, und nach links (d.h. er wird dekrementiert), wenn Slots „zurückgegeben“ werden. Wenn ein Slot „entnommen“ wird, wird in der Liste die Konstante `NO_SLOT` geschrieben. Alle freien Postfächer sind „rechts“ des Indexes (und an der Stelle des Indexes selbst) verzeichnet (vgl. Abb. 3.3).

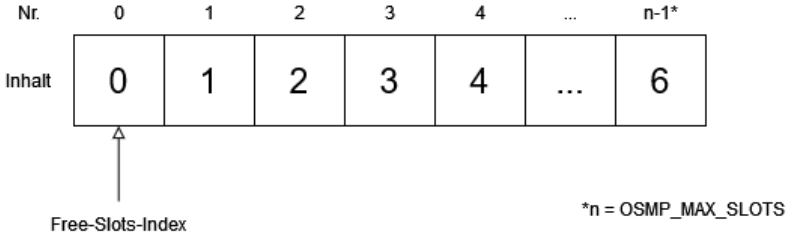


Abbildung 3.2: Die Free-Slots-Liste unmittelbar nach der Initialisierung des Shared Memory.

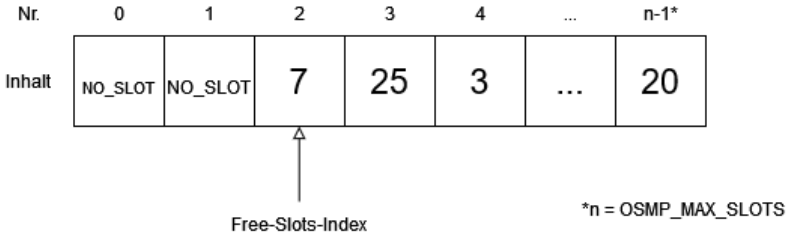


Abbildung 3.3: Ein fiktiver Zustand der Free-Slots-Liste im Laufe des Programms.

3.1.2 Funktionalität der Postfächer

Das Postfach (vgl. den unteren linken Teil von Abb. 3.1 auf S. 4) ist als Ring angelegt. Der in- und der out-Index zeigen initial auf die gleiche Stelle. Sobald per in-Index eine Stelle belegt wird, wird der in-Index inkrementiert. Der out-Index indiziert aber noch

die Stelle, die gerade belegt wurde; diese enthält nun den Verweis auf einen Slot, aus dem gelesen werden kann. Nicht belegte Fächer im Postfach werden auf die Konstante `NO_MESSAGE` gesetzt.

3.2 OSMP-Funktionen

3.2.1 Blockierende Funktionen

OSMP_Send

Bei `OSMP_Send()` wird geprüft, ob der empfangene Prozess schon gestartet und fertig initialisiert wurde. Wenn nicht, wird gewartet. Falls der empfangende Prozess voll mit Nachrichten ist, wird mit Hilfe eines Semaphore gewartet. Außerdem wird ggfs. wird auf einen freien Slot mit Hilfe eines Semaphore gewartet, falls es keine freie Slots vorhanden sind. In den freien Slot wird die Nachricht geschrieben und die Adresse der Nachricht wird in das Postfach des empfangenen Prozesses geschrieben, so kann der Prozess später die Nachricht lesen. Es wird dazu mit Hilfe von einem Semaphore signalisiert, dass der Prozess eine Nachricht empfangen kann. Folgender Pseudocode fasst die Synchronisierung von `OSMP_Send()` zusammen. Für eine bessere Übersicht sind die unterschiedlichen Zugriffe auf Mutexe und Semaphoren farblich gekennzeichnet. So wird im Vergleich mit `OSMP_Recv` (3.2.1) ersichtlich, dass Schreiben und Lesen nahezu symmetrisch zueinander funktionieren.

```
1:  semwait(sem_proc_empty)          ▷ freier Platz im Postfach (Ring)
2:  semwait(sem_shm_free_slots       ▷ freier Nachrichtenslot
3:  mutex_lock(mutex_shm_free_slots
4:  entnimm Slot aus Liste
5:  mutex_unlock(mutex_shm_free_slots
6:  schreibe Nachricht in Slot
7:  mutex_lock(mutex_proc_in)
8:  lies in-Index
9:  schreibe Slot in Postfach
10: hole nächsten Index
11: mutex_unlock(mutex_proc_in)
12: semsignal(sem_proc_full)
```

OSMP_Recv

Bei OSMP_Recv wird zunächst mit Hilfe eines Semaphore gewartet, bis eine Nachricht da ist. Es wird ein Index von der nächsten Nachricht im Postfach-Ring (vgl. 3.1.2) benutzt, um herauszufinden, welcher der Slot der nächsten Nachricht ist. Dann wird diese

Nachricht in den Buffer des empfangenen Prozesses kopiert und mit Hilfe von eines Semaphore wird signalisiert, dass es wieder einen freien Platz für Nachrichten gibt. Analog zu Abschnitt 3.2.1 fasst der folgende Pseudocode das Schreiben zusammen:

```
1: semwait(sem_proc_full)
2: mutex_lock(mutex_proc_out)
3: lies out-Index
4: lies Postfach (Index für Slot), lösche Index
5: inkrementiere out-Index
6: mutex_unlock(mutex_proc_out)
7: semsignal(sem_proc_empty)
8: kopiere Nachricht in User-Space
9: mutex_lock(mutex_shm_free_slots)
10: schreibe freien Slot in Liste
11: mutex_unlock(mutex_shm_free_slots)
12: semsignal(sem_shm_free_slots)
```

OSMP_Barrier

Bei `OSMP_Barrier()` wird ein Counter vom Anfangswert OSMP_Size runter gezählt, bis ein Prozess null erreicht. Mögliche Race-Conditions werden mit Hilfe eines Mutex verhindert. Wenn ein Prozess nicht der letzte ist, der die Barriere, wird durch eine Condition Variable gewartet, bis der Zyklus der Barriere erhöht wird. Um *spurious wakeups* zu verhindern, wird das Warten in einer while-Schleife mit einem Prädikat (Prüfung, ob Zyklus immer noch der gleiche ist) durchgeführt. Der letzte Prozess muss nicht mehr warten; er erhöht den Zyklus und signalisiert mit Hilfe eines Broadcasts, dass es für alle weitergehen kann. In Pseudocode ausgedrückt:

```
1: prüfe, ob Barrier initialisiert ist (wenn nicht: Fehler)
2: mutex_lock(barrier_mutex)
3: speichere aktuellen Zyklus
4: dekrementiere Counter                    ▷ User-Threads ausschließen!
5: if letzter Prozess then
6:     re-initialisiere Barrier für nächsten Durchlauf und benachrichtige alle wartenden
       Prozesse per Condition-Variable
7: else
8:     warte an Condition-Variable, bis Zyklusnummer nicht mehr der gespeicherten
       Zyklusnummer entspricht
9: mutex_unlock(barrier_mutex)
```

OSMP_Gather

Bei `OSMP_Gather()` schreibt jeder Prozess eine Nachricht in seinen Gather-Slot. Dann wird mit Hilfe von `OSMP_Barrier()` (s.o.) blockiert, bis alle Prozesse ihre Nachricht in ihren Slot geschrieben haben. So wird sichergestellt, dass der Prozess, der die Nachrichten sammelt, nicht frühzeitig anfängt, sie zu lesen. Wenn der Root (der empfangende Prozess) anfängt zu lesen, werden alle andere mit Hilfe vom Barrier 3.2.1 nochmal blockiert, so können wir gewährleisten, dass es erst weitergeht, wenn der Root das Lesen beendet hat, sodass keine Probleme mit Race Conditions auftreten. In Pseudocode ausgedrückt:

```
1: kopiere eigene zu sendende Nachricht in den eigenen Gather-Slot
2: warte, bis alle Prozesse in ihren eigenen Gather-Slot geschrieben haben (Barrier)
3: if empfangender Prozess (Root) then
4:     mutex_lock(gather_mutex)
5:     kopiere Nachrichten aus allen Gather-Slots in User-Space
6:     mutex_unlock(gather_mutex)
7: warte, bis Root gelesen hat (Barrier)
```

3.2.2 Nicht blockierende Funktionen

OSMP_ISend

In `OSMP_ISend()` wird das Struct IParams(s.u.) benutzt, um die asychrone Funktionalität des Sendens zu ermöglichen. Am Anfang werden alle Parameter in das Struct geschrieben. Dann wird ein Thread zu einer Linked List hinzugefügt, um später alle gestarteten Threads zu verwalten. Dieser Thread startet asynchron das Senden der Nachricht mit Hilfe von OSMP_Send (s.o.). Wenn der Thread fertig ist, wird das Flag im IParams-Struct auf `OSMP_DONE` gesetzt, um dem Aufrufer signalisieren zu können, dass das Senden abgeschlossen ist. Er muss dies aber selbstständig mit `OSMP_Test()` (nicht blockierend) oder `OSMP_Wait()` (blockierend) prüfen.

OSMP_IRecv

In `OSMP_IRecv()` wird analog zu `OSMP_ISend()` das Struct IParams (s.u.) benutzt, um die asychrone Funktionalität des Empfangens zu ermöglichen. Am Anfang werden alle Parameter in das Struct geschrieben. Dann wird ein Thread zu einer Linked List hinzugefügt, um später alle gestarteten Threads zu verwalten. Dieser Thread startet asynchron das Empfangen der Nachricht mit Hilfe von `OSMP_Recv` (s.o.). Wenn der Thread fertig ist, wird das Flag im IParams-Struct auf `OSMP_DONE` gesetzt, um dem Aufrufer signalisieren zu können, dass das Empfangen abgeschlossen ist. Er muss dies aber selbstständig mit `OSMP_Test()` (nicht blockierend) oder `OSMP_Wait()` (blockierend) prüfen.

IParams

IParams (vgl. Abb. 3.4) hat alle Attribute, um Send/Recv zu ermöglichen sowie die Attribute für die synchronisation zwischen den Threads. Dazu zählt ein Mutex für den Zugriff auf die Elemente des Structs, eine Condition-Variable, um auf das Fertigstellen des mit diesem Struct assoziierten Prozesses zu warten (in `OSMP_Wait()`), und done, dass signalisiert, ob der mit diesem Struct assoziierte blockierende Vorgang abgeschlossen ist.

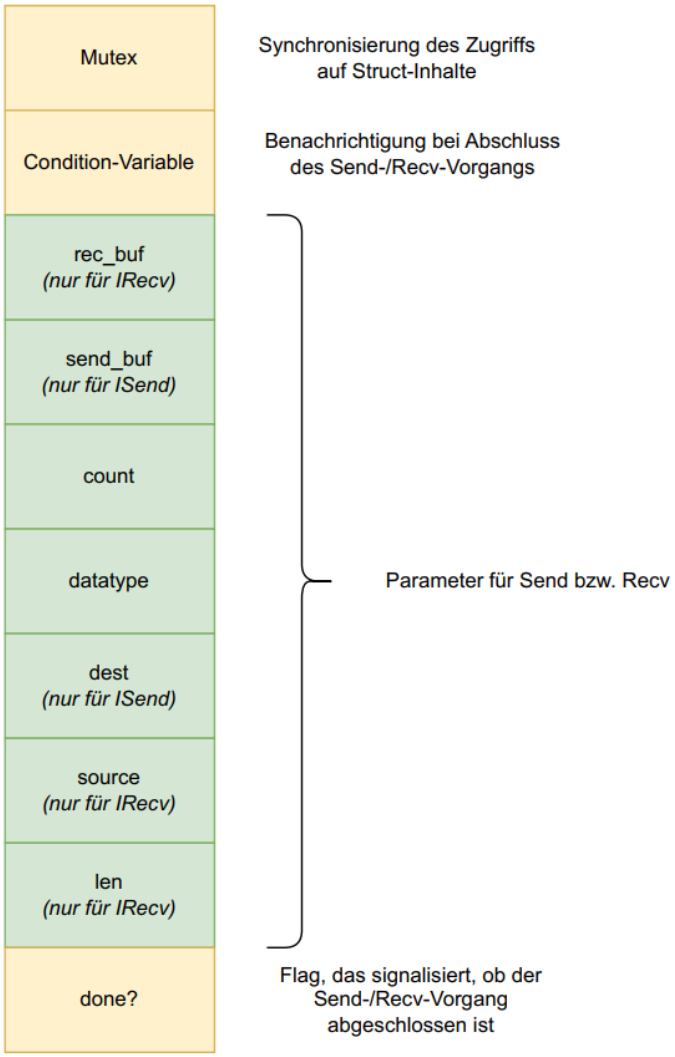


Abbildung 3.4: Die Struktur der IParams.

3.3 Logger

Der Logger bietet die Funktionalität abhängig von der Verbosität zu loggen. Die Verbosität beträgt drei Stufen (vgl. Praktikumsbeschreibung) Beim Loggen wird die Zeitpunkt zusammen mit der PID gelogt. Es kann eine Log-Datei beim Starten des Runner gewählt werden. Wenn es keine gewählt wird, wird es in log.log geloggt. Die Dateiname wird im Shared Memory gespeichert und lokal beim Initialsieren kopiert. Am Ende des Programms wird mit free sicher gestellt, dass alle Resourcen freigegeben werden. Das Loggen wird mit einem Logging-Mutex gemacht, um zu gewährleisten, dass keine Probleme vom Typ Race Conditions auftreten. Nach dem erfolgreichen Lock des Mutex wird die Datei mit `fopen()` und `append` geöffnet, dann wird mit `fprintf()` in die Datei geschrieben und am Ende mit `fclose()` geschlossen.

OSMP - Entwurf und Implementierung einer Message Passing Umgebung für Interprozesskommunikation

Erstellt am 13.06.2024.

Erzeugt von Doxygen 1.9.1

Kapitel 1

Klassen-Verzeichnis

1.1 Auflistung der Klassen

Hier folgt die Aufzählung aller Klassen, Strukturen, Varianten und Schnittstellen mit einer Kurzbeschreibung:

Kapitel 2

Datei-Verzeichnis

2.1 Auflistung der Dateien

Hier folgt die Aufzählung aller Dateien mit einer Kurzbeschreibung:

Kapitel 3

Klassen-Dokumentation

3.1 barrier_t Strukturreferenz

Datentyp zur Beschreibung einer Barriere.

`#include <osmplib.h>`

Öffentliche Attribute

- pthread_mutex_t mutex
- pthread_cond_t convar
- int valid
- int counter
- int cycle

3.1.1 Ausführliche Beschreibung

Datentyp zur Beschreibung einer Barriere.

3.1.2 Dokumentation der Datenelemente

3.1.2.1 convar

`pthread_cond_t barrier_t::convar`

3.1.2.2 counter

```
int barrier_t::counter
```

3.1.2.3 cycle

```
int barrier_t::cycle
```

3.1.2.4 mutex

```
pthread_mutex_t barrier_t::mutex
```

3.1.2.5 valid

```
int barrier_t::valid
```

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

- src/osmp_library/osmplib.h

3.2 IParams Strukturreferenz

Struct, das die ISend-/IRecv-Funktionsparameter speichert,.

```
#include <osmplib.h>
```

Öffentliche Attribute

- pthread_mutex_t mutex
- pthread_cond_t convar
- void * recv_buf
- const void * send_buf
- int count
- OSMP_Datatype datatype
- int dest
- int * source
- int * len
- int done

3.2.1 Ausführliche Beschreibung

Struct, das die ISend-/IRecv-Funktionsparameter speichert,.

3.2.2 Dokumentation der Datenelemente

3.2.2.1 convar

`IParams::convar`

Condition-Variable, um auf das Fertigstellen des mit diesem Struct assoziierten Prozesses zu warten.

3.2.2.2 count

`IParams::count`

Übergabeparameter für Send/Recv, der hier zwischengespeichert wird.

3.2.2.3 datatype

`IParams::datatype`

Übergabeparameter für Send/Recv, der hier zwischengespeichert wird.

3.2.2.4 dest

`IParams::dest`

Übergabeparameter für Send, der hier zwischengespeichert wird.

3.2.2.5 done

`IParams::done`

Flag, das signalisiert, ob der mit diesem Struct assoziierte blockierende Vorgang abgeschlossen ist. Steht auf *OSMP_DONE*, wenn abgeschlossen, andernfalls auf *OSMP_WAITING*.

3.2.2.6 len

`IParams::len`

Übergabeparameter für Recv, der hier zwischengespeichert wird.

3.2.2.7 mutex

`IParams::mutex`

Mutex für den Zugriff auf die Elemente des Structs.

3.2.2.8 recv_buf

`IParams::recv_buf`

Übergabeparameter für Recv, der hier zwischengespeichert wird.

3.2.2.9 send_buf

`IParams::send_buf`

Übergabeparameter für Recv, der hier zwischengespeichert wird.

3.2.2.10 source

`IParams::source`

Übergabeparameter für Recv, der hier zwischengespeichert wird.

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

- src/osmp_library/osmplib.h

3.3 message_slot Strukturreferenz

Struct für eine Nachricht entsprechend der Definition unseres Shared Memory.

`#include <osmplib.h>`

Öffentliche Attribute

- int from
- int len
- OSMP_Datatype type
- char payload [OSMP_MAX_PAYLOAD_LENGTH]

3.3.1 Ausführliche Beschreibung

Struct für eine Nachricht entsprechend der Definition unseres Shared Memory.

3.3.2 Dokumentation der Datenelemente

3.3.2.1 from

`message_slot::from`

Rang des sendenden Prozesses.

3.3.2.2 len

`message_slot::len`

Länge der Nachricht in Bytes.

3.3.2.3 payload

`message_slot::payload`

Inhalt der Nachricht.

3.3.2.4 type

`message_slot::type`

Datentyp der enthaltenen Nachricht

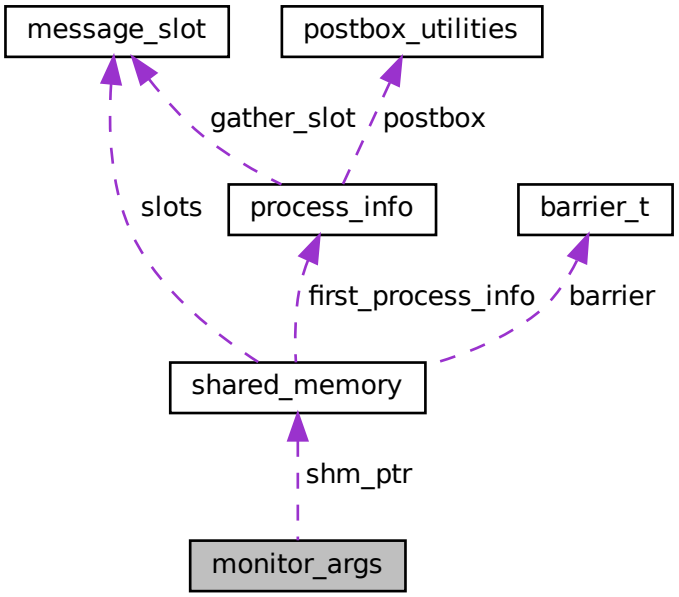
Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

- src/osmp_library/osmplib.h

3.4 monitor_args Strukturreferenz

`#include <osmp_run.h>`

Zusammengehörigkeiten von monitor_args:



Öffentliche Attribute

- int number_of_executables
- int flag
- int shared_memory_fd
- shared_memory * shm_ptr

3.4.1 Dokumentation der Datenelemente

3.4.1.1 flag

`int monitor_args::flag`

3.4.1.2 number_of_executables

```
int monitor_args::number_of_executables
```

3.4.1.3 shared_memory_fd

```
int monitor_args::shared_memory_fd
```

3.4.1.4 shm_ptr

```
shared_memory* monitor_args::shm_ptr
```

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

- src/osmp_runner/osmp_run.h

3.5 postbox_utilities Strukturreferenz

```
#include <osmplib.h>
```

Öffentliche Attribute

- int postbox [OSMP_MAX_MESSAGES_PROC]
- int in_index
- pthread_mutex_t mutex_proc_in
- int out_index
- pthread_mutex_t mutex_proc_out
- sem_t sem_proc_empty
- sem_t sem_proc_full
- int sem_proc_full_value

3.5.1 Dokumentation der Datenelemente

3.5.1.1 in_index

```
postbox_utilities::in_index
```

Index, der auf den nächsten freien Platz im Postfach zeigt.

3.5.1.2 mutex_proc_in

`postbox_utilities::mutex_proc_in`

Mutex für die Synchronisierung der in_index-Variable.

3.5.1.3 mutex_proc_out

`postbox_utilities::mutex_proc_out`

Mutex für die Synchronisierung der out_index-Variable.

3.5.1.4 out_index

`postbox_utilities::out_index`

Index, der auf die nächste Nachricht im Postfach zeigt.

3.5.1.5 postbox

`postbox_utilities::postbox`

Array, das alle Nachrichten für den Prozess enthält (als Index des Nachrichtenslots).

3.5.1.6 sem_proc_empty

`postbox_utilities::sem_proc_empty`

Semaphore für freie Plätze im Postfach.

3.5.1.7 sem_proc_full

`postbox_utilities::sem_proc_full`

Semaphore für belegte Plätze im Postfach.

3.5.1.8 sem_proc_full_value

`int postbox_utilities::sem_proc_full_value`

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

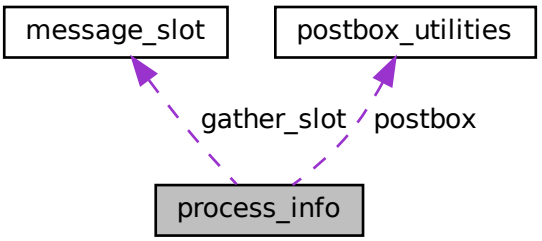
- src/osmp_library/osmplib.h

3.6 process_info Strukturreferenz

Struct für Informationen zu einem Prozess.

`#include <osmplib.h>`

Zusammengehörigkeiten von process_info:



Öffentliche Attribute

- int rank
- int pid
- postbox_utilities postbox
- message_slot gather_slot
- int available

3.6.1 Ausführliche Beschreibung

Struct für Informationen zu einem Prozess.

3.6.2 Dokumentation der Datenelemente

3.6.2.1 available

`process_info::available`

Ein Flag um zu wiesen, ob dieser Prozess verfügbar ist.

3.6.2.2 gather_slot

`process_info::gather_slot`

Der Gather-Slot des Prozesses.

3.6.2.3 pid

`process_info::pid`

PID des Prozesses.

3.6.2.4 postbox

`process_info::postbox`

Postfach des Prozesses.

3.6.2.5 rank

`process_info::rank`

Rang des Prozesses.

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

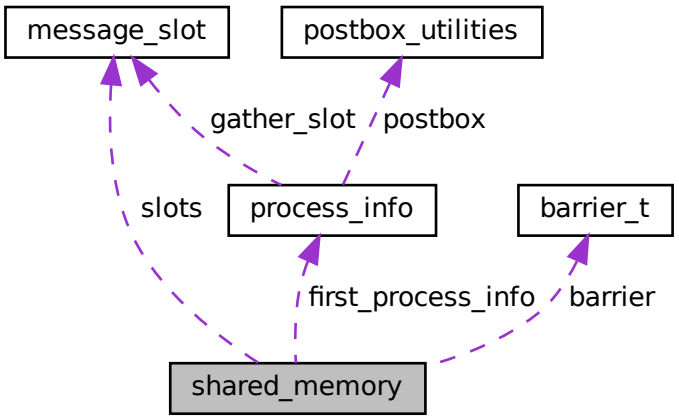
- src/osmp_library/osmplib.h

3.7 shared_memory Strukturreferenz

Struct für den fixen Teil des Shared Memory gemäß unserer Spezifikation.

`#include <osmplib.h>`

Zusammengehörigkeiten von shared_memory:



Öffentliche Attribute

- int size
- pthread_mutex_t logging_mutex
- pthread_mutex_t initializing_mutex
- pthread_cond_t initializing_condition
- int free_slots [OSMP_MAX_SLOTS]
- int free_slots_index
- sem_t sem_shm_free_slots
- pthread_mutex_t mutex_shm_free_slots
- message_slot slots [OSMP_MAX_SLOTS]
- pthread_mutex_t gather_mutex
- barrier_t barrier
- char logfile [MAX_PATH_LENGTH]
- unsigned int verbosity
- process_info first_process_info

3.7.1 Ausführliche Beschreibung

Struct für den fixen Teil des Shared Memory gemäß unserer Spezifikation.

3.7.2 Dokumentation der Datenelemente

3.7.2.1 barrier

```
shared_memory::barrier
```

Barriere.

3.7.2.2 first_process_info

```
process_info shared_memory::first_process_info
```

3.7.2.3 free_slots

```
shared_memory::free_slots
```

Liste der freien Nachrichtenslots.

3.7.2.4 free_slots_index

```
shared_memory::free_slots_index
```

Zeigt auf die Stelle in free_slots, an der das nächste freie Postfach liegt.

3.7.2.5 gather_mutex

`shared_memory::gather_mutex`

Mutex für den Zugriff auf alle Gather-Slots durch ein und denselben Prozess (lesender Gather-Root-Prozess).

3.7.2.6 initializing_condition

`shared_memory::initializing_condition`

Ein Condition für das Warten bis ein Prozess initialisiert würde.

3.7.2.7 initializing_mutex

`shared_memory::initializing_mutex`

Mutex für die Initialisierung von den OSMP_Prozessen.

3.7.2.8 logfile

`shared_memory::logfile`

Pfad zur Logdatei.

3.7.2.9 logging_mutex

`shared_memory::logging_mutex`

Mutex für den Zugriff auf die Logdatei.

3.7.2.10 mutex_shm_free_slots

`shared_memory::mutex_shm_free_slots`

Mutex zur Synchronisierung des Zugriffs auf die Liste der freien Nachrichtenslots und deren Index.

3.7.2.11 sem_shm_free_slots

`shared_memory::sem_shm_free_slots`

Semaphore für die Vergabe von Nachrichtenslots.

3.7.2.12 size

`shared_memory::size`

Anzahl der Kindprozesse (entspricht OSMP_Size).

3.7.2.13 slots

`shared_memory::slots`

Array mit allen 1:1-Nachrichtenslots.

3.7.2.14 verbosity

`shared_memory::verbosity`

Logging-Verbosität.

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

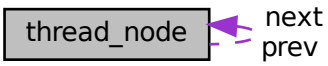
- src/osmp_library/osmplib.h

3.8 thread_node Strukturreferenz

Eine two way linked list von threads,.

`#include <osmplib.h>`

Zusammengehörigkeiten von thread_node:



Öffentliche Attribute

- pthread_t thread

  Der Thread in diesem Knoten.
- struct thread_node * next

  Ein Pointer auf dem nächsten Knoten in der Liste, wenn es der letzte Element ist, dann zeigt es auf Null.
- struct thread_node * prev

3.8.1 Ausführliche Beschreibung

Eine two way linked list von threads,.

3.8.2 Dokumentation der Datenelemente

3.8.2.1 next

`thread_node::next`

Ein Pointer auf dem nächsten Knoten in der Liste, wenn es der letzte Element ist, dann zeigt es auf Null.

Ein Pointer auf dem letzten Knoten in der Liste, wenn es der erste Element ist, dann zeigt es auf Null.

3.8.2.2 prev

`struct thread_node* thread_node::prev`

3.8.2.3 thread

`thread_node::thread`

Der Thread in diesem Knoten.

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

- src/osmp_library/osmplib.h

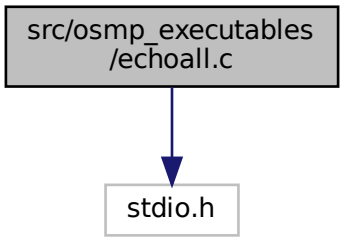
Kapitel 4

Datei-Dokumentation

4.1 src/osmp_executables/echoall.c-Dateireferenz

```
#include <stdio.h>
```

Include-Abhängigkeitsdiagramm für echoall.c:



Funktionen

- int main (int argc, char ∗argv[ ])

4.1.1 Dokumentation der Funktionen

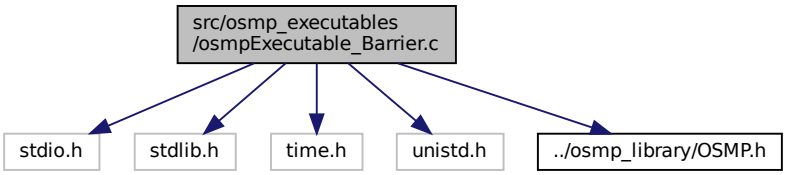
4.1.1.1 main()

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

4.2 src/osmp_executables/ospExecutable_Barrier.c-Dateireferenz

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <unistd.h>
#include "../osmp_library/OSMP.h"
```

Include-Abhängigkeitsdiagramm für ospExecutable_Barrier.c:



Funktionen

- int main (int argc, char ∗argv[ ])

4.2.1 Dokumentation der Funktionen

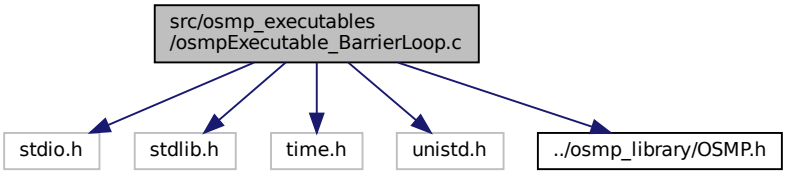
4.2.1.1 main()

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

4.3 src/osmp_executables/ospExecutable_BarrierLoop.c-Dateireferenz

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <unistd.h>
#include "../osmp_library/OSMP.h"
```

Include-Abhängigkeitsdiagramm für ospExecutable_BarrierLoop.c:



Funktionen

- int main (int argc, char ∗argv[ ])

4.3.1 Dokumentation der Funktionen

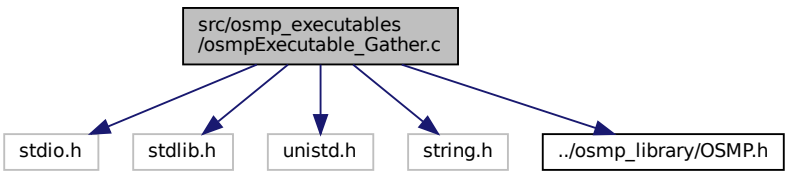
4.3.1.1 main()

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

4.4 src/osmp_executables/ospExecutable_Gather.c-Dateireferenz

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include "../osmp_library/OSMP.h"
```

Include-Abhängigkeitsdiagramm für ospExecutable_Gather.c:



Funktionen

- int main (int argc, char ∗argv[ ])

4.4.1 Dokumentation der Funktionen

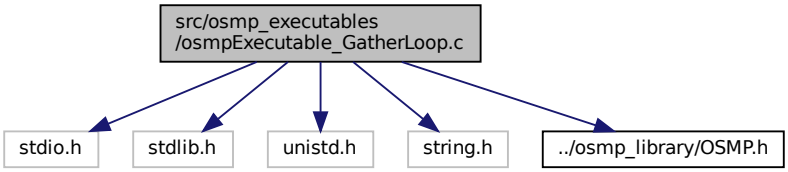
4.4.1.1 main()

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

4.5 src/osmp_executables/ospExecutable_GatherLoop.c-Dateireferenz

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include "../osmp_library/OSMP.h"
```

Include-Abhängigkeitsdiagramm für ospExecutable_GatherLoop.c:



Makrodefinitionen

- #define LOOPS 100

Funktionen

- int main (int argc, char ∗argv[ ])

4.5.1 Makro-Dokumentation

4.5.1.1 LOOPS

```
#define LOOPS 100
```

4.5.2 Dokumentation der Funktionen

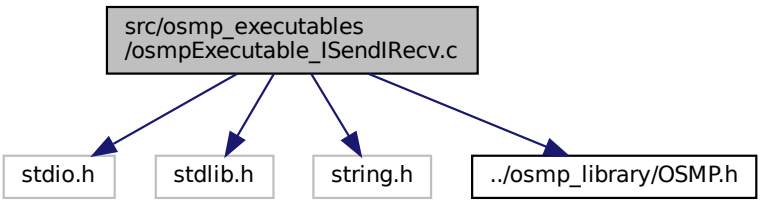
4.5.2.1 main()

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

4.6 src/osmp_executables/ospExecutable_ISendIRecv.c-Dateireferenz

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "../osmp_library/OSMP.h"
```

Include-Abhängigkeitsdiagramm für ospExecutable_ISendIRecv.c:



Funktionen

- int main (int argc, char ∗argv[ ])

4.6.1 Dokumentation der Funktionen

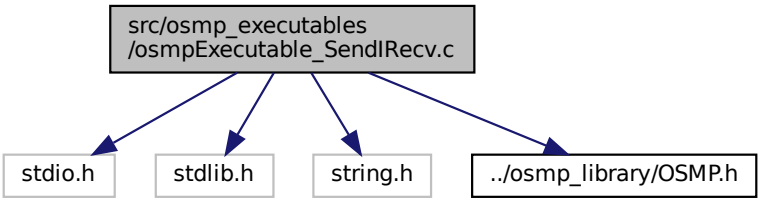
4.6.1.1 main()

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

4.7 src/osmp_executables/ospExecutable_SendIRecv.c-Dateireferenz

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

```
#include "../osmp_library/OSMP.h"
```
Include-Abhängigkeitsdiagramm für ompExecutable_SendIRecv.c:



Funktionen

- int main (int argc, char ∗argv[ ])

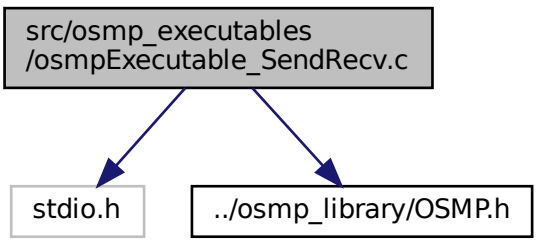
4.7.1 Dokumentation der Funktionen

4.7.1.1 main()

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

4.8 src/osmp_executables/ompExecutable_SendRecv.c-Dateireferenz

```
#include <stdio.h>
#include "../osmp_library/OSMP.h"
```
Include-Abhängigkeitsdiagramm für ompExecutable_SendRecv.c:



Funktionen

- int main (int argc, char ∗argv[ ])

4.8.1 Dokumentation der Funktionen

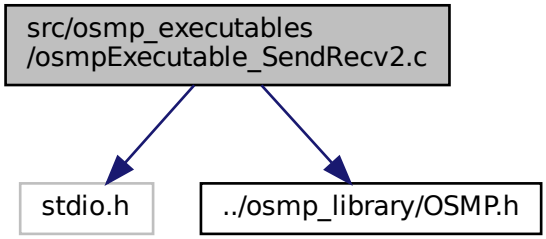
4.8.1.1 main()

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

4.9 src/osmp_executables/ospExecutable_SendRecv2.c-Dateireferenz

```
#include <stdio.h>
#include "../osmp_library/OSMP.h"
```

Include-Abhängigkeitsdiagramm für ospExecutable_SendRecv2.c:



Funktionen

- int main (int argc, char ∗argv[ ])

4.9.1 Dokumentation der Funktionen

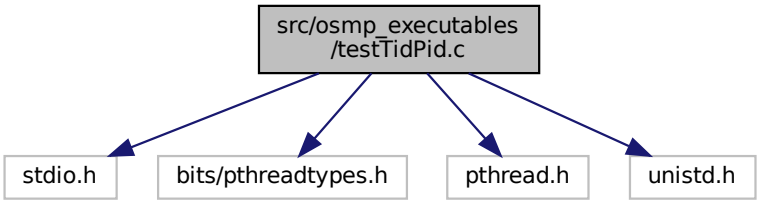
4.9.1.1 main()

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

4.10 src/osmp_executables/testTidPid.c-Dateireferenz

```
#include <stdio.h>
#include <bits/pthreadtypes.h>
#include <pthread.h>
#include <unistd.h>
```

Include-Abhängigkeitsdiagramm für testTidPid.c:



Makrodefinitionen

- #define _GNU_SOURCE

Funktionen

- void * check (void *args)
- int main (void)

4.10.1 Makro-Dokumentation

4.10.1.1 _GNU_SOURCE

```
#define _GNU_SOURCE
```

Kleines Programm, mit dem gezeigt werden kann, dass die PID bei einem Prozess und seinem Kindthread gleich ist, die TID aber nicht.

4.10.2 Dokumentation der Funktionen

4.10.2.1 check()

```
void* check (
            void * args )
```

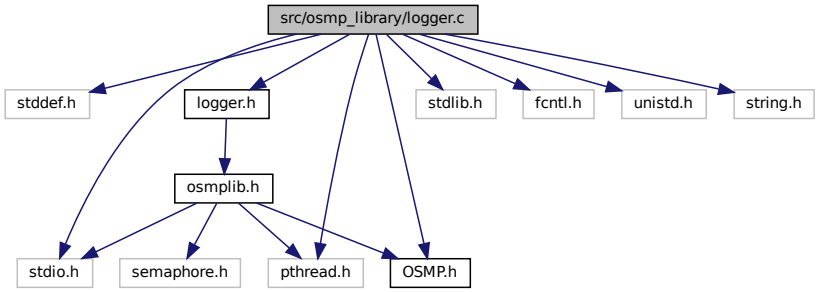
4.10.2.2 main()

```
int main (
            void  )
```

4.11 src/osmp_library/logger.c-Dateireferenz

```
#include <stddef.h>
#include <stdio.h>
#include <pthread.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#include <string.h>
#include "logger.h"
#include "OSMP.h"
```

Include-Abhängigkeitsdiagramm für logger.c:



Funktionen

- void init_file (const char *filename)
- void logging_init_parent (shared_memory *shm, char *name, int log_verbosity)
- void logging_init_child (shared_memory *shm)
- void log_to_file (int level, char *message)
- void logging_close (void)
- char * get_logfile_name (void)

Variablen

- pthread_mutex_t * mutex
- FILE * logging_file
- char * file_name = NULL
- int verbosity = 1

4.11.1 Dokumentation der Funktionen

4.11.1.1 get_logfile_name()

```
char* get_logfile_name (
            void  )
```

Gibt den Dateinamen zum Loggen zurück.

Rückgabe

Der Name der Datei als String.

4.11.1.2 init_file()

```
void init_file (
            const char * filename )
```

Öffnet die angegebene Logdatei und leert sie. Falls sie nicht existiert, wird sie neu erstellt.

Parameter

filename	Zeiger auf den Dateipfad der Logdatei.

4.11.1.3 log_to_file()

```
void log_to_file (
            int level,
            char * message )
```

Schreibt in die Logdatei.

Parameter

level	Logging-Level des Eintrags (1-3).
message	Zu loggende Nachricht.

4.11.1.4 logging_close()

```
void logging_close (
            void  )
```

Schließe das Logging ab. Die Logdatei wird geschlossen.

4.11.1.5 logging_init_child()

```
void logging_init_child (
            shared_memory * shm )
```

Initialisiert die Log-Bibliothek für das Kind. Es speichert die verbosität und Dateiname.

Parameter

shared_memory	Pointer auf den Shared Memory
memory_size	Größe des Shared Memory in Bytes

4.11.1.6 logging_init_parent()

```
void logging_init_parent (
            shared_memory * shm,
            char * name,
            int log_verbosity )
```

Initialisiert die Log-Bibliothek. Erzeugt Logdatei, falls sie noch nicht existiert. Diese Methode ist für das Elter, das Kind hat eine eigene Methode.

Parameter

shm	Pointer auf den Shared Memory.
name	Pfad zur Logdatei.
log_verbosity	Logging-Verbosität (Level 1-3). Bei ungültigem Wert wird das Standard-Level 1 verwendet.

4.11.2 Variablen-Dokumentation

4.11.2.1 file_name

```
char* file_name = NULL
```

4.11.2.2 logging_file

```
FILE* logging_file
```

4.11.2.3 mutex

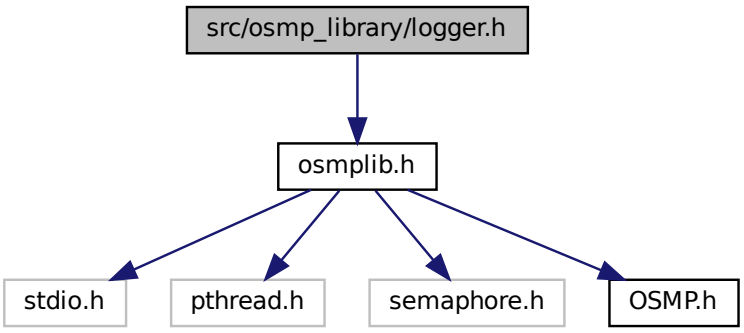
```
pthread_mutex_t* mutex
```

4.11.2.4 verbosity

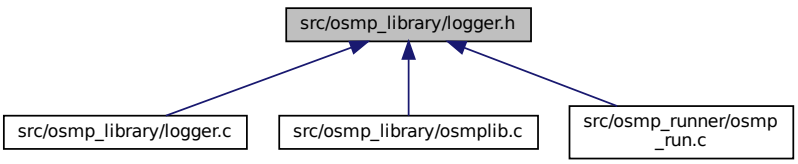
```
int verbosity = 1
```

4.12 src/osmp_library/logger.h-Dateireferenz

```
#include "osmplib.h"
```
Include-Abhängigkeitsdiagramm für logger.h:



Dieser Graph zeigt, welche Datei direkt oder indirekt diese Datei enthält:



Funktionen

- void logging_init_parent (shared_memory *shm, char *name, int log_verbosity)
- void log_to_file (int level, char *message)
- void logging_close (void)
- char * get_logfile_name (void)
- void logging_init_child (shared_memory *shm)

4.12.1 Dokumentation der Funktionen

4.12.1.1 get_logfile_name()

```
char* get_logfile_name (
            void  )
```

Gibt den Dateinamen zum Loggen zurück.

Rückgabe

Der Name der Datei als String.

4.12.1.2 log_to_file()

```
void log_to_file (
            int level,
            char * message )
```

Schreibt in die Logdatei.

Parameter

level	Logging-Level des Eintrags (1-3).
message	Zu loggende Nachricht.

4.12.1.3 logging_close()

```
void logging_close (
            void  )
```

Schließe das Logging ab. Die Logdatei wird geschlossen.

4.12.1.4 logging_init_child()

```
void logging_init_child (
            shared_memory * shm )
```

Initialisiert die Log-Bibliothek für das Kind. Es speichert die verbosität und Dateiname.

Parameter

shared_memory	Pointer auf den Shared Memory
memory_size	Größe des Shared Memory in Bytes

4.12.1.5 logging_init_parent()

```
void logging_init_parent (
            shared_memory * shm,
            char * name,
            int log_verbosity )
```

Initialisiert die Log-Bibliothek. Erzeugt Logdatei, falls sie noch nicht existiert. Diese Methode ist für das Elter, das Kind hat eine eigene Methode.

Parameter

shm	Pointer auf den Shared Memory.
name	Pfad zur Logdatei.
log_verbosity	Logging-Verbosität (Level 1-3). Bei ungültigem Wert wird das Standard-Level 1 verwendet.

4.13 src/osmp_library/OSMP.h-Dateireferenz

Dieser Graph zeigt, welche Datei direkt oder indirekt diese Datei enthält:

Makrodefinitionen

- #define OSMP_SUCCESS 0
- #define OSMP_FAILURE (!OSMP_SUCCESS)
- #define OSMP_MAX_MESSAGES_PROC 16
- #define OSMP_MAX_SLOTS 256
- #define OSMP_MAX_PAYLOAD_LENGTH 1024
- #define OSMP_DONE 1
- #define OSMP_WAITING 0

Typdefinitionen

- typedef void * OSMP_Request
- typedef enum OSMP_Datatype OSMP_Datatype

Aufzählungen

- enum OSMP_Datatype {
  OSMP_SHORT , OSMP_INT , OSMP_LONG , OSMP_UNSIGNED_CHAR ,
  OSMP_UNSIGNED , OSMP_UNSIGNED_SHORT , OSMP_UNSIGNED_LONG , OSMP_FLOAT ,
  OSMP_DOUBLE , OSMP_BYTE }

Funktionen

- int get_OSMP_MAX_PAYLOAD_LENGTH (void)
- int get_OSMP_MAX_SLOTS (void)
- int get_OSMP_MAX_MESSAGES_PROC (void)
- int get_OSMP_FAILURE (void)
- int get_OSMP_SUCCESS (void)
- int OSMP_SizeOf (OSMP_Datatype datatype, unsigned int *size)
- int OSMP_Init (const int *argc, char ***argv)
- int OSMP_Size (int *size)
- int OSMP_Rank (int *rank)
- int OSMP_Send (const void *buf, int count, OSMP_Datatype datatype, int dest)
- int OSMP_Recv (void *buf, int count, OSMP_Datatype datatype, int *source, int *len)
- int OSMP_Finalize (void)
- int OSMP_Barrier (void)
- int OSMP_Gather (void *sendbuf, int sendcount, OSMP_Datatype sendtype, void *recvbuf, int recvcount, OSMP_Datatype recvtype, int root)
- int OSMP_ISend (const void *buf, int count, OSMP_Datatype datatype, int dest, OSMP_Request request)
- int OSMP_IRecv (void *buf, int count, OSMP_Datatype datatype, int *source, int *len, OSMP_Request request)
- int OSMP_Test (OSMP_Request request, int *flag)
- int OSMP_Wait (OSMP_Request request)
- int OSMP_CreateRequest (OSMP_Request *request)
- int OSMP_RemoveRequest (OSMP_Request *request)
- int OSMP_GetSharedMemoryName (char **name)
- void OSMP_GetSharedMemoryPointer (char **shared_memory)

4.13.1 Makro-Dokumentation

4.13.1.1 OSMP_DONE

```
#define OSMP_DONE 1
```

Gibt an, dass eine nicht-blockierende Funktion (ISend/IRecv) abgeschlossen ist.

4.13.1.2 OSMP_FAILURE

```
#define OSMP_FAILURE ( !OSMP_SUCCESS )
```

Im Fehlerfall liefern die OSMP-Funktionen den Wert OSMP_FAILURE zurück. Die Fehler führen aber nicht zum beenden des Programms (z. B. wenn ein Prozess eine Nachricht an einen nicht existierenden Prozess schickt).

4.13.1.3 OSMP_MAX_MESSAGES_PROC

```
#define OSMP_MAX_MESSAGES_PROC 16
```

Die maximale Zahl der Nachrichten pro Prozess

4.13.1.4 OSMP_MAX_PAYLOAD_LENGTH

```
#define OSMP_MAX_PAYLOAD_LENGTH 1024
```

Die maximale Länge der Nutzlast einer Nachricht

4.13.1.5 OSMP_MAX_SLOTS

```
#define OSMP_MAX_SLOTS 256
```

Die maximale Anzahl der Nachrichten, die insgesamt vorhanden sein dürfen

4.13.1.6 OSMP_SUCCESS

```
#define OSMP_SUCCESS 0
```

Alle OSMP-Funktionen liefern im Erfolgsfall OSMP_SUCCESS als Rückgabewert. Weitere Rückgabewerte können mit Begründung (und Dokumentation!) definiert werden

4.13.1.7 OSMP_WAITING

```
#define OSMP_WAITING 0
```

Gibt an, dass eine nicht-blockierende Funktion (ISend/IRecv) noch nicht abgeschlossen ist.

4.13.2 Dokumentation der benutzerdefinierten Typen

4.13.2.1 OSMP_Datatype

```
typedef enum OSMP_Datatype OSMP_Datatype
```

Die OSMP-Datentypen entsprechen den C-Datentypen. Sie werden verwendet, um den Typ der Daten anzugeben, die mit den OSMP-Funktionen gesendet bzw. empfangen werden sollen.

4.13.2.2 OSMP_Request

```
typedef void* OSMP_Request
```

4.13.3 Dokumentation der Aufzählungstypen

4.13.3.1 OSMP_Datatype

```
enum OSMP_Datatype
```

Die OSMP-Datentypen entsprechen den C-Datentypen. Sie werden verwendet, um den Typ der Daten anzugeben, die mit den OSMP-Funktionen gesendet bzw. empfangen werden sollen.

Aufzählungswerte

OSMP_SHORT	
OSMP_INT	
OSMP_LONG	
OSMP_UNSIGNED_CHAR	
OSMP_UNSIGNED	
OSMP_UNSIGNED_SHORT	
OSMP_UNSIGNED_LONG	
OSMP_FLOAT	
OSMP_DOUBLE	
OSMP_BYTE	

4.13.4 Dokumentation der Funktionen

4.13.4.1 get_OSMP_FAILURE()

```
int get_OSMP_FAILURE (
            void  )
```

Gibt den Wert von OSMP_FAILURE zurück.

4.13.4.2 get_OSMP_MAX_MESSAGES_PROC()

```
int get_OSMP_MAX_MESSAGES_PROC (
            void  )
```

Gibt die maximale Zahl der Nachrichten pro Prozess zurück.

4.13.4.3 get_OSMP_MAX_PAYLOAD_LENGTH()

```
int get_OSMP_MAX_PAYLOAD_LENGTH (
            void  )
```

Gibt die maximale Länge der Nutzlast einer Nachricht zurück.

4.13.4.4 get_OSMP_MAX_SLOTS()

```
int get_OSMP_MAX_SLOTS (
            void  )
```

Gibt die Maximale Anzahl der Nachrichten, die insgesamt vorhanden sein dürfen zurück.

4.13.4.5 get_OSMP_SUCCESS()

```
int get_OSMP_SUCCESS (
            void  )
```

Gibt den Wert von OSMP_SUCCESS zurück.

4.13.4.6 OSMP_Barrier()

```
int OSMP_Barrier (
            void  )
```

Diese kollektive Funktion blockiert den aufrufenden Prozess. Erst wenn alle anderen Prozesse ebenfalls an der Barriere angekommen sind, laufen die Prozesse weiter.

Rückgabe

Im Erfolgsfall OSMP_SUCCESS, sonst OSMP_FAILURE

4.13.4.7 OSMP_CreateRequest()

```
int OSMP_CreateRequest (
            OSMP_Request * request )
```

Erstellt eine OSMP_Request. Eine OSMP_Request wird dazu verwendet, um nicht blockierende Operationen zu überwachen.

Parameter

out	*request*	Adresse eines Requests (input)

Rückgabe

Im Erfolgsfall OSMP_SUCCESS, sonst OSMP_FAILURE

4.13.4.8 OSMP_Finalize()

```
int OSMP_Finalize (
            void  )
```

Alle OSMP-Prozesse müssen diese Funktion aufrufen, bevor sie sich beenden. Sie geben damit den Zugriff auf die gemeinsamen Ressourcen frei. Hierbei muss jeder Prozess zuvor alle noch vorhandenen Nachrichten abarbeiten. Dies bedeutet, dass der Posteingang gesperrt wird und alle noch vorhandenen Nachrichten gelöscht werden.

Rückgabe

Im Erfolgsfall OSMP_SUCCESS, sonst OSMP_FAILURE

4.13.4.9 OSMP_Gather()

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

Diese Funktion ermöglicht die Gather-Kommunikation. Alle Prozesse müssen die Funktion aufrufen. Es muss sicher gestellt, dass der Empfangspuffer die Größe des Sendepuffers von alle Prozesse zusammen entspricht. Hierbei können mehrere Prozesse an einen Empfänger Prozess Daten schicken. Die Prozesse werden blockiert bis der Empfänger bis Ende die Daten liest.

Parameter

in	*sendbuf*	Zeiger auf den Sendepuffer.
in	*sendcount*	Anzahl der Elemente im Sendepuffer.
in	*sendtype*	OSMP-Datentyp der Elemente im Sendepuffer.
out	*recvbuf*	Zeiger auf den Empfangspuffer.
in	*recvcount*	Anzahl der Elemente im Empfangspuffer.
in	*recvtype*	OSMP-Datentyp der Elemente im Empfangspuffer.
in	*root*	Rang des empfangenden Prozesses.

Rückgabe

Im Erfolgsfall OSMP_SUCCESS, sonst OSMP_FAILURE

4.13.4.10 OSMP_GetSharedMemoryName()

```
int OSMP_GetSharedMemoryName (
            char ** name )
```

Gibt den Namen des Shared Memory Bereichs zurück.

Parameter

out	*name*	Der Name des Shared Memory Bereichs

Rückgabe

Im Erfolgsfall OSMP_SUCCESS, sonst OSMP_FAILURE

4.13.4.11 OSMP_GetSharedMemoryPointer()

```
void OSMP_GetSharedMemoryPointer (
            char ** shared_memory )
```

4.13.4.12 OSMP_Init()

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

Die Funktion OSMP_Init() initialisiert die OSMP-Umgebung und ermöglicht den Zugang zu den gemeinsamen Ressourcen der OSMP-Prozesse. Sie muss von jedem OSMP-Prozess zu Beginn aufgerufen werden. Durch diesen Aufruf wird außerdem der Posteingang des Prozesses freigegeben.

Parameter

in	*argc*	Adresse der Argumentzahl
in	*argv*	Adresse des Argumentvektors

Rückgabe

Im Erfolgsfall OSMP_SUCCESS, sonst OSMP_FAILURE

4.13.4.13 OSMP_IRecv()

```
int OSMP_IRecv (
            void * buf,
            int count,
            OSMP_Datatype datatype,
            int * source,
            int * len,
            OSMP_Request request )
```

Die Funktion empfängt eine Nachricht analog zu OSMP_Recv(). Die Funktion kehrt jedoch sofort zurück, ohne dass das Kopieren der Nachricht sichergestellt ist (nicht blockierendes Empfangen).

Parameter

out	*buf*	Startadresse des Speicherbereichs, wo die zu empfangende Nachricht gespeichert werden soll.
in	*count*	Zahl der Elemente vom angegebenen Typ, die empfangen werden können
in	*datatype*	OSMP-Typ der Daten im Puffer
out	*source*	PID des Senders zwischen 0, . . . , np-1
out	*len*	tatsächliche Länge der empfangenen Nachricht in Byte
in,out	*request*	Adresse einer Datenstruktur, die später verwendet werden kann, um abzufragen, ob die die Operation abgeschlossen ist.

Rückgabe

Im Erfolgsfall OSMP_SUCCESS, sonst OSMP_FAILURE

4.13.4.14 OSMP_ISend()

```
int OSMP_ISend (
            const void * buf,
            int count,
            OSMP_Datatype datatype,
            int dest,
            OSMP_Request request )
```

Die Funktion sendet eine Nachricht analog zu OSMP_Send(). Die Funktion kehrt jedoch sofort zurück, ohne dass das Kopieren der Nachricht sichergestellt ist (nicht blockierendes Senden).

Parameter

in	*buf*	Startadresse des Puffers mit der zu sendenden Nachricht
in	*count*	Zahl der Elemente vom angegebenen Typ im Puffer
in	*datatype*	OSMP-Typ der Daten im Puffer
in	*dest*	PID des Empfängers zwischen 0, . . . , np-1
in,out	*request*	Adresse einer eigenen Datenstruktur, die später verwendet werden kann, um abzufragen, ob die Operation abgeschlossen ist.

Rückgabe

Im Erfolgsfall OSMP_SUCCESS, sonst OSMP_FAILURE

4.13.4.15 OSMP_Rank()

```
int OSMP_Rank (
            int * rank )
```

Die Funktion OSMP_Rank() liefert in ∗rank die OSMP-Prozessnummer des aufrufenden OSMP-Prozesses von 0,. . . ,np-1 zurück.

Parameter

out	*rank*	Prozessnummer 0,. . . ,np-1 des aktuellen OSMP-Prozesse

Rückgabe

Im Erfolgsfall OSMP_SUCCESS, sonst OSMP_FAILURE

4.13.4.16 OSMP_Recv()

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

Der aufrufende Prozess empfängt eine Nachricht mit maximal count Elementen des angegebenen Datentyps datatype. Die Nachricht wird an die Adresse buf des aufrufenden Prozesses geschrieben. Unter source wird die OSMP-Prozessnummer des sendenden Prozesses und unter len die tatsächliche Länge der gelesenen Nachricht abgelegt. Die Funktion ist blockierend, d.h. sie wartet, bis eine Nachricht für den Prozess vorhanden ist. Wenn die Funktion zurückkehrt, ist der Kopierprozess abgeschlossen. Die Nachricht gilt nach dem Aufruf dieser Funktion als abgearbeitet.

Parameter

out	*buf*	Startadresse des Puffers im lokalen Speicher des aufrufenden Prozesses, in den die Nachricht kopiert werden soll.
in	*count*	maximale Zahl der Elemente vom angegebenen Typ, die empfangen werden können
in	*datatype*	OSMP-Typ der Daten im Puffer
out	*source*	Nummer des Senders zwischen 0,. . . ,np-1
out	*len*	tatsächliche Länge der empfangenen Nachricht in Byte

Rückgabe

Im Erfolgsfall OSMP_SUCCESS, sonst OSMP_FAILURE

4.13.4.17 OSMP_RemoveRequest()

```
int OSMP_RemoveRequest (
            OSMP_Request * request )
```

Löscht eine OSMP_Request.

Parameter

in	*request*	Adresse eines Requests

Rückgabe

Im Erfolgsfall OSMP_SUCCESS, sonst OSMP_FAILURE

4.13.4.18 OSMP_Send()

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

Die Funktion OSMP_Send() sendet eine Nachricht an den Prozess mit der Nummer dest. Die Nachricht besteht aus count Elementen vom Typ datatype. Die zu sendende Nachricht beginnt im aufrufenden Prozess bei der Adresse buf. Die Funktion ist blockierend, d.h. wenn sie in das aufrufende Programm zurückkehrt, ist der Kopiervorgang abgeschlossen.

Parameter

in	*buf*	Startadresse des Puffers mit der zu sendenden Nachricht
in	*count*	Zahl der Elemente vom angegebenen Typ im Puffer
in	*datatype*	OSMP-Typ der Daten im Puffer
in	*dest*	Nummer des Empfängers zwischen 0,. . . ,np-1

Rückgabe

Im Erfolgsfall OSMP_SUCCESS, sonst OSMP_FAILURE

4.13.4.19 OSMP_Size()

```
int OSMP_Size (
            int * size )
```

Die Funktion OSMP_Size() liefert in *size* die Zahl der OSMP-Prozesse ohne den OSMP-Starter Prozess zurück. Sollte mit der Zahl übereinstimmen, die in der Kommandozeile dem OSMP-Starter übergeben wird.

Parameter

out	*size*	Zahl der OSMP-Prozesse

Rückgabe

Im Erfolgsfall OSMP_SUCCESS, sonst OSMP_FAILURE

4.13.4.20 OSMP_SizeOf()

```
int OSMP_SizeOf (
            OSMP_Datatype datatype,
            unsigned int * size )
```

Die Funktion OSMP_SizeOf() liefert in *size* die Größe des Datentyps datatype in Byte zurück.

Parameter

in	*datatype*	OSMP-Datentyp
out	*size*	Größe des Datentyps in Byte

Rückgabe

Im Erfolgsfall OSMP_SUCCESS; falls der OSMP_Datatype nicht existiert, OSMP_FAILURE

4.13.4.21 OSMP_Test()

```
int OSMP_Test (
            OSMP_Request request,
            int * flag )
```

Die Funktion testet, ob die mit der Request verknüpften Operation abgeschlossen ist. Sie ist nicht blockierend, d.h. sie wartet nicht auf das Ende der mit request verknüpften Operation.

Parameter

in	*request*	Adresse der Struktur, die eine blockierende Operation spezifiziert
out	*flag*	Gibt den Status der Operation an.

Rückgabe

Im Erfolgsfall OSMP_SUCCESS, sonst OSMP_FAILURE

4.13.4.22 OSMP_Wait()

```
int OSMP_Wait (
            OSMP_Request request )
```

Die Funktion wartet, bis die mit der Request verknüpfte, nicht blockierende Operation abgeschlossen ist. Sie ist so lange blockiert, bis dies der Fall ist.

Parameter

in	*request*	Adresse der Struktur, die eine nicht blockierende Operation spezifiziert

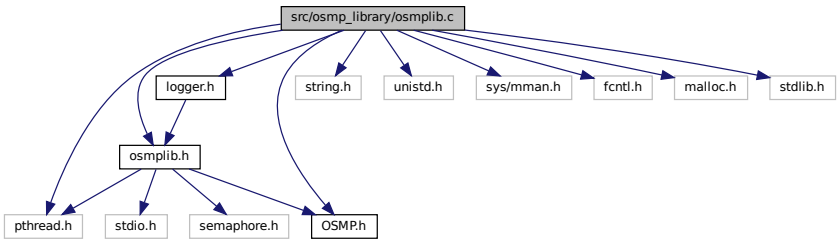
Rückgabe

Im Erfolgsfall OSMP_SUCCESS, sonst OSMP_FAILURE

4.14 src/osmp_library/osmplib.c-Dateireferenz

```
#include "osmplib.h"
#include "logger.h"
#include "OSMP.h"
#include <string.h>
#include <unistd.h>
#include <sys/mman.h>
#include <pthread.h>
#include <fcntl.h>
#include <malloc.h>
#include <stdlib.h>
```

Include-Abhängigkeitsdiagramm für osmplib.c:



Makrodefinitionen

- #define SHARED_MEMORY_NAME "/shared_memory"
- #define _GNU_SOURCE

Funktionen

- void log_osmp_lib_call (const char *function_name)
- process_info * get_process_info (int rank)
- int get_next_message (void)
- int create_thread (pthread_t **thread)
- void wait_and_finalize_all_threads (void)
- void OSMP_Init_Runner (int fd, shared_memory *shm, int size)
- int calculate_shared_memory_size (int processes)
- int barrier_wait (barrier_t *barrier)
- int OSMP_Init (const int *argc, char ***argv)
- int OSMP_SizeOf (OSMP_Datatype datatype, unsigned int *size)
- int OSMP_Size (int *size)
- int OSMP_Rank (int *rank)
- int OSMP_Send (const void *buf, int count, OSMP_Datatype datatype, int dest)
- int OSMP_Recv (void *buf, int count, OSMP_Datatype datatype, int *source, int *len)
- int OSMP_Finalize (void)
- int OSMP_Barrier (void)
- int OSMP_Gather (void *sendbuf, int sendcount, OSMP_Datatype sendtype, void *recvbuf, int recvcount, OSMP_Datatype recvtype, int root)
- void * OSMP_thread_send (void *args)
- int OSMP_ISend (const void *buf, int count, OSMP_Datatype datatype, int dest, OSMP_Request request)
- void * OSMP_thread_recv (void *args)
- int OSMP_IRecv (void *buf, int count, OSMP_Datatype datatype, int *source, int *len, OSMP_Request request)
- int OSMP_Test (OSMP_Request request, int *flag)
- int OSMP_Wait (OSMP_Request request)
- int OSMP_CreateRequest (OSMP_Request *request)
- int OSMP_RemoveRequest (OSMP_Request *request)
- int OSMP_GetSharedMemoryName (char **name)

Variablen

- shared_memory * shm_ptr = NULL
- int shared_memory_fd
- int OSMP_size
- int OSMP_rank = OSMP_FAILURE
- int memory_size
- thread_node * erster_thread = NULL
- thread_node * letzter_thread = NULL
- pthread_mutex_t * thread_linked_list_mutex = NULL

4.14.1 Makro-Dokumentation

4.14.1.1 _GNU_SOURCE

```
#define _GNU_SOURCE
```

4.14.1.2 SHARED_MEMORY_NAME

```
#define SHARED_MEMORY_NAME "/shared_memory"
```

In dieser Quelltext-Datei sind Implementierungen der OSMP Bibliothek zu finden.

4.14.2 Dokumentation der Funktionen

4.14.2.1 barrier_wait()

```
int barrier_wait (
            barrier_t * barrier )
```

Interne Implementierung der OSMP_Barrier()-Funktion. Der Hauptteil wird mithilfe der Posix-Funktion pthread_↩ cond_wait() implementiert, die eine Voraussetzung (Prediction) und eine Bedingung (Condition) benötigt.

Parameter

barrier	Zeiger auf die Barriere, an der gewartet werden soll.

Rückgabe

Im Erfolgsfall OSMP_SUCCESS, sonst OSMP_FAILURE.

4.14.2.2 calculate_shared_memory_size()

```
int calculate_shared_memory_size (
            int processes )
```

Berechnet den für den Shared Memory benötigten Speicherplatz in Abhängigkeit von der Anzahl der Prozesse.

Parameter

processes	Die Anzahl der Executable-Prozesse, die verwaltet werden.

Rückgabe

Die Größe des benötigten Speicherplatzes in Bytes.

4.14.2.3 create_thread()

```
int create_thread (
            pthread_t ** thread )
```

Die Funktion create_thread() fügt ein neues Thread in die liste der Threads und speichert den in thread.

Parameter

thread	Der gestarte Thread.

Rückgabe

Im Erfolgsfall OSMP_SUCCESS, sonst OSMP_FAILURE

4.14.2.4 get_next_message()

```
int get_next_message (
            void  )
```

Gibt den Index des Nachrichtenslots zurück, in dem die nächste Nachricht für den aufrufenden Prozess liegt.

Rückgabe

Index des Slots, in dem die nächste Nachricht für den aufrufenden Prozess liegt.

4.14.2.5 get_process_info()

```
process_info* get_process_info (
            int rank )
```

Gibt einen Zeiger auf das process_info-Struct des angegebenen Prozesses zurück.

Parameter

rank	Rang des Prozesses, dessen process_info angefordert wird.

Rückgabe

Zeiger auf process_info-Struct des Prozesses mit dem angegebenen Rang.

4.14.2.6 log_osmp_lib_call()

```
void log_osmp_lib_call (
            const char * function_name )
```

Übergibt eine Level-1-Lognachricht an den Logger.

Parameter

pid	Die Process ID des aufrufenden Prozesses.
function_name	Der Name der aufrufenden Funktion.

4.14.2.7 OSMP_Barrier()

```
int OSMP_Barrier (
            void  )
```

Diese kollektive Funktion blockiert den aufrufenden Prozess. Erst wenn alle anderen Prozesse ebenfalls an der Barriere angekommen sind, laufen die Prozesse weiter.

Rückgabe

Im Erfolgsfall OSMP_SUCCESS, sonst OSMP_FAILURE

4.14.2.8 OSMP_CreateRequest()

```
int OSMP_CreateRequest (
            OSMP_Request * request )
```

Erstellt eine OSMP_Request. Eine OSMP_Request wird dazu verwendet, um nicht blockierende Operationen zu überwachen.

Parameter

out	*request*	Adresse eines Requests (input)

Rückgabe

Im Erfolgsfall OSMP_SUCCESS, sonst OSMP_FAILURE

4.14.2.9 OSMP_Finalize()

```
int OSMP_Finalize (
            void  )
```

Alle OSMP-Prozesse müssen diese Funktion aufrufen, bevor sie sich beenden. Sie geben damit den Zugriff auf die gemeinsamen Ressourcen frei. Hierbei muss jeder Prozess zuvor alle noch vorhandenen Nachrichten abarbeiten. Dies bedeutet, dass der Posteingang gesperrt wird und alle noch vorhandenen Nachrichten gelöscht werden.

Rückgabe

Im Erfolgsfall OSMP_SUCCESS, sonst OSMP_FAILURE

4.14.2.10 OSMP_Gather()

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

Diese Funktion ermöglicht die Gather-Kommunikation. Alle Prozesse müssen die Funktion aufrufen. Es muss sicher gestellt, dass der Empfangspuffer die Größe des Sendepuffers von alle Prozesse zusammen entspricht. Hierbei können mehrere Prozesse an einen Empfänger Prozess Daten schicken. Die Prozesse werden blockiert bis der Empfänger bis Ende die Daten liest.

Parameter

in	*sendbuf*	Zeiger auf den Sendepuffer.
in	*sendcount*	Anzahl der Elemente im Sendepuffer.
in	*sendtype*	OSMP-Datentyp der Elemente im Sendepuffer.
out	*recvbuf*	Zeiger auf den Empfangspuffer.
in	*recvcount*	Anzahl der Elemente im Empfangspuffer.
in	*recvtype*	OSMP-Datentyp der Elemente im Empfangspuffer.
in	*root*	Rang des empfangenden Prozesses.

Rückgabe

Im Erfolgsfall OSMP_SUCCESS, sonst OSMP_FAILURE

4.14.2.11 OSMP_GetSharedMemoryName()

```
int OSMP_GetSharedMemoryName (
            char ** name )
```

Gibt den Namen des Shared Memory Bereichs zurück.

Parameter

out	*name*	Der Name des Shared Memory Bereichs

Rückgabe

Im Erfolgsfall OSMP_SUCCESS, sonst OSMP_FAILURE

4.14.2.12 OSMP_Init()

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

Die Funktion OSMP_Init() initialisiert die OSMP-Umgebung und ermöglicht den Zugang zu den gemeinsamen Ressourcen der OSMP-Prozesse. Sie muss von jedem OSMP-Prozess zu Beginn aufgerufen werden. Durch diesen Aufruf wird außerdem der Posteingang des Prozesses freigegeben.

Parameter

in	*argc*	Adresse der Argumentzahl
in	*argv*	Adresse des Argumentvektors

Rückgabe

Im Erfolgsfall OSMP_SUCCESS, sonst OSMP_FAILURE

4.14.2.13 OSMP_Init_Runner()

```
void OSMP_Init_Runner (
            int fd,
            shared_memory * shm,
            int size )
```

Setzt die globalen Variablen der OSMP-Bibliothek für den Elternprozess.

Parameter

fd	Shared-Memory-File-Descriptor.
shm	Zeiger auf den Beginn des Shared Memory.
size	Größe des Shared Memory in Bytes.

4.14.2.14 OSMP_IRecv()

```
int OSMP_IRecv (
            void * buf,
            int count,
            OSMP_Datatype datatype,
            int * source,
            int * len,
            OSMP_Request request )
```

Die Funktion empfängt eine Nachricht analog zu OSMP_Recv(). Die Funktion kehrt jedoch sofort zurück, ohne dass das Kopieren der Nachricht sichergestellt ist (nicht blockierendes Empfangen).

Parameter

out	*buf*	Startadresse des Speicherbereichs, wo die zu empfangende Nachricht gespeichert werden soll.
in	*count*	Zahl der Elemente vom angegebenen Typ, die empfangen werden können
in	*datatype*	OSMP-Typ der Daten im Puffer
out	*source*	PID des Senders zwischen 0, . . . , np-1
out	*len*	tatsächliche Länge der empfangenen Nachricht in Byte
in,out	*request*	Adresse einer Datenstruktur, die später verwendet werden kann, um abzufragen, ob die die Operation abgeschlossen ist.

Rückgabe

Im Erfolgsfall OSMP_SUCCESS, sonst OSMP_FAILURE

4.14.2.15 OSMP_ISend()

```
int OSMP_ISend (
            const void * buf,
            int count,
            OSMP_Datatype datatype,
            int dest,
            OSMP_Request request )
```

Die Funktion sendet eine Nachricht analog zu OSMP_Send(). Die Funktion kehrt jedoch sofort zurück, ohne dass das Kopieren der Nachricht sichergestellt ist (nicht blockierendes Senden).

Parameter

in	*buf*	Startadresse des Puffers mit der zu sendenden Nachricht
in	*count*	Zahl der Elemente vom angegebenen Typ im Puffer
in	*datatype*	OSMP-Typ der Daten im Puffer
in	*dest*	PID des Empfängers zwischen 0, . . . , np-1
in,out	*request*	Adresse einer eigenen Datenstruktur, die später verwendet werden kann, um abzufragen, ob die Operation abgeschlossen ist.

Rückgabe

Im Erfolgsfall OSMP_SUCCESS, sonst OSMP_FAILURE

4.14.2.16 OSMP_Rank()

```
int OSMP_Rank (
            int * rank )
```

Die Funktion OSMP_Rank() liefert in ∗rank die OSMP-Prozessnummer des aufrufenden OSMP-Prozesses von 0,. . . ,np-1 zurück.

Parameter

out	*rank*	Prozessnummer 0,. . . ,np-1 des aktuellen OSMP-Prozesse

Rückgabe

Im Erfolgsfall OSMP_SUCCESS, sonst OSMP_FAILURE

4.14.2.17 OSMP_Recv()

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

Der aufrufende Prozess empfängt eine Nachricht mit maximal count Elementen des angegebenen Datentyps datatype. Die Nachricht wird an die Adresse buf des aufrufenden Prozesses geschrieben. Unter source wird die OSMP-Prozessnummer des sendenden Prozesses und unter len die tatsächliche Länge der gelesenen Nachricht abgelegt. Die Funktion ist blockierend, d.h. sie wartet, bis eine Nachricht für den Prozess vorhanden ist. Wenn die Funktion zurückkehrt, ist der Kopierprozess abgeschlossen. Die Nachricht gilt nach dem Aufruf dieser Funktion als abgearbeitet.

Parameter

out	*buf*	Startadresse des Puffers im lokalen Speicher des aufrufenden Prozesses, in den die Nachricht kopiert werden soll.
in	*count*	maximale Zahl der Elemente vom angegebenen Typ, die empfangen werden können
in	*datatype*	OSMP-Typ der Daten im Puffer
out	*source*	Nummer des Senders zwischen 0,. . . ,np-1
out	*len*	tatsächliche Länge der empfangenen Nachricht in Byte

Rückgabe

Im Erfolgsfall OSMP_SUCCESS, sonst OSMP_FAILURE

4.14.2.18 OSMP_RemoveRequest()

```
int OSMP_RemoveRequest (
            OSMP_Request * request )
```

Löscht eine OSMP_Request.

Parameter

in	*request*	Adresse eines Requests

Rückgabe

Im Erfolgsfall OSMP_SUCCESS, sonst OSMP_FAILURE

4.14.2.19 OSMP_Send()

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

Die Funktion OSMP_Send() sendet eine Nachricht an den Prozess mit der Nummer dest. Die Nachricht besteht aus count Elementen vom Typ datatype. Die zu sendende Nachricht beginnt im aufrufenden Prozess bei der Adresse buf. Die Funktion ist blockierend, d.h. wenn sie in das aufrufende Programm zurückkehrt, ist der Kopiervorgang abgeschlossen.

Parameter

in	*buf*	Startadresse des Puffers mit der zu sendenden Nachricht
in	*count*	Zahl der Elemente vom angegebenen Typ im Puffer
in	*datatype*	OSMP-Typ der Daten im Puffer
in	*dest*	Nummer des Empfängers zwischen 0,...,np-1

Rückgabe

Im Erfolgsfall OSMP_SUCCESS, sonst OSMP_FAILURE

4.14.2.20 OSMP_Size()

```
int OSMP_Size (
            int * size )
```

Die Funktion OSMP_Size() liefert in *size* die Zahl der OSMP-Prozesse ohne den OSMP-Starter Prozess zurück. Sollte mit der Zahl übereinstimmen, die in der Kommandozeile dem OSMP-Starter übergeben wird.

Parameter

out	*size*	Zahl der OSMP-Prozesse

Rückgabe

Im Erfolgsfall OSMP_SUCCESS, sonst OSMP_FAILURE

4.14.2.21 OSMP_SizeOf()

```
int OSMP_SizeOf (
            OSMP_Datatype datatype,
            unsigned int * size )
```

Die Funktion OSMP_SizeOf() liefert in *size* die Größe des Datentyps datatype in Byte zurück.

Parameter

in	*datatype*	OSMP-Datentyp
out	*size*	Größe des Datentyps in Byte

Rückgabe

Im Erfolgsfall OSMP_SUCCESS; falls der OSMP_Datatype nicht existiert, OSMP_FAILURE

4.14.2.22 OSMP_Test()

```
int OSMP_Test (
            OSMP_Request request,
            int * flag )
```

Die Funktion testet, ob die mit der Request verknüpften Operation abgeschlossen ist. Sie ist nicht blockierend, d.h. sie wartet nicht auf das Ende der mit request verknüpften Operation.

Parameter

in	*request*	Adresse der Struktur, die eine blockierende Operation spezifiziert
out	*flag*	Gibt den Status der Operation an.

Rückgabe

Im Erfolgsfall OSMP_SUCCESS, sonst OSMP_FAILURE

4.14.2.23 OSMP_thread_recv()

```
void* OSMP_thread_recv (
            void * args )
```

4.14.2.24 OSMP_thread_send()

```
void* OSMP_thread_send (
            void * args )
```

Asynchron starten vom send durch einen Thread.

Parameter

args	die Argumente für den thread.

Rückgabe

Im Erfolgsfall Null, sonst OSMP_FAILURE

4.14.2.25 OSMP_Wait()

```
int OSMP_Wait (
            OSMP_Request request )
```

Die Funktion wartet, bis die mit der Request verknüpfte, nicht blockierende Operation abgeschlossen ist. Sie ist so lange blockiert, bis dies der Fall ist.

Parameter

in	*request*	Adresse der Struktur, die eine nicht blockierende Operation spezifiziert

Rückgabe

Im Erfolgsfall OSMP_SUCCESS, sonst OSMP_FAILURE

4.14.2.26 wait_and_finalize_all_threads()

```
void wait_and_finalize_all_threads (
            void  )
```

Wartet bis alle gestartete Threads fertig sind und beendet die.

4.14.3 Variablen-Dokumentation

4.14.3.1 erster_thread

```
thread_node* erster_thread = NULL
```

4.14.3.2 letzter_thread

```
thread_node* letzter_thread = NULL
```

4.14.3.3 memory_size

```
int memory_size
```

4.14.3.4 OSMP_rank

```
int OSMP_rank = OSMP_FAILURE
```

4.14.3.5 OSMP_size

```
int OSMP_size
```

4.14.3.6 shared_memory_fd

```
int shared_memory_fd
```

4.14.3.7 shm_ptr

```
shared_memory* shm_ptr = NULL
```

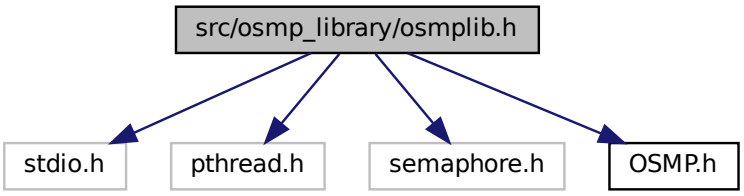
4.14.3.8 thread_linked_list_mutex

```
pthread_mutex_t* thread_linked_list_mutex = NULL
```

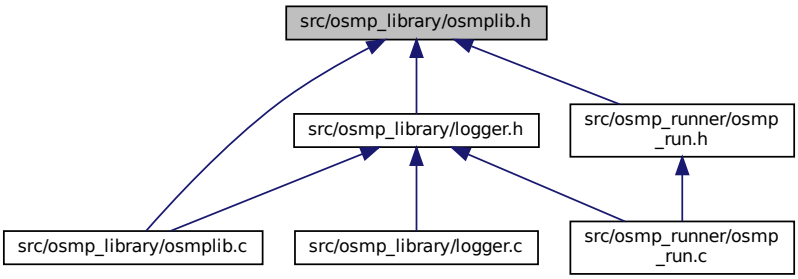
4.15 src/osmp_library/osmplib.h-Dateireferenz

```
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#include "OSMP.h"
```

Include-Abhängigkeitsdiagramm für osmplib.h:



Dieser Graph zeigt, welche Datei direkt oder indirekt diese Datei enthält:



Klassen

- struct message_slot

  Struct für eine Nachricht entsprechend der Definition unseres Shared Memory.
- struct postbox_utilities
- struct process_info

  Struct für Informationen zu einem Prozess.
- struct barrier_t

  Datentyp zur Beschreibung einer Barriere.
- struct shared_memory

  Struct für den fixen Teil des Shared Memory gemäß unserer Spezifikation.
- struct thread_node

  Eine two way linked list von threads,.
- struct IParams

  Struct, das die ISend-/IRecv-Funktionsparameter speichert,.

Makrodefinitionen

- #define UNUSED(x) { (void)(x); }
- #define NO_MESSAGE (-1)
- #define NO_SLOT (-1)
- #define SLOT_FREE 1
- #define SLOT_TAKEN 0
- #define BARRIER_VALID 1
- #define NOT_SAVED 1
- #define SAVED 0
- #define NOT_AVAILABLE 1
- #define AVAILABLE 0
- #define MAX_PATH_LENGTH 256

Typdefinitionen

- typedef struct message_slot message_slot
- typedef struct process_info process_info
- typedef struct barrier_t barrier_t
- typedef struct shared_memory shared_memory
- typedef struct thread_node thread_node
- typedef struct IParams IParams

Funktionen

- int calculate_shared_memory_size (int processes)
- void OSMP_Init_Runner (int fd, shared_memory ∗shm, int size)
- process_info ∗ get_process_info (int rank)

4.15.1 Makro-Dokumentation

4.15.1.1 AVAILABLE

```
#define AVAILABLE 0
```

Flag, um singlanisieren, dass der Prozess erreichbar ist.

4.15.1.2 BARRIER_VALID

```
#define BARRIER_VALID 1
```

Flag, um eine korrekt initialisierte Barrier zu kennzeichnen.

4.15.1.3 MAX_PATH_LENGTH

```
#define MAX_PATH_LENGTH 256
```

Maximal erlaubte Länge des Pfads zur Logdatei, inkl. terminierendem Nullbyte.

4.15.1.4 NO_MESSAGE

```
#define NO_MESSAGE (-1)
```

Mit diesem Wert wird signalisiert, dass ein Prozess keine weitere Nachricht hat.

4.15.1.5 NO_SLOT

```
#define NO_SLOT (-1)
```

Mit diesem Wert werden alle nicht genutzten Elemente des Arrays belegt, in dem die freien Nachrichtenslots verzeichnet sind.

4.15.1.6 NOT_AVAILABLE

```
#define NOT_AVAILABLE 1
```

Flag, um singlanisieren, dass der Prozess nicht erreichbar ist.

4.15.1.7 NOT_SAVED

```
#define NOT_SAVED 1
```

Flag, um singlanisieren, dass der Reciever nicht alle Nachrichten in Gather gespeichert hat.

4.15.1.8 SAVED

```
#define SAVED 0
```

Flag, um singlanisieren, dass der Reciever alle Nachrichten in Gather gespeichert hat.

4.15.1.9 SLOT_FREE

```
#define SLOT_FREE 1
```

Flag für einen freien Nachrichtenslot.

4.15.1.10 SLOT_TAKEN

```
#define SLOT_TAKEN 0
```

Flag für einen belegten Nachrichtenslot.

4.15.1.11 UNUSED

```
#define UNUSED(
              x ) { (void)(x); }
```

Dieses Makro wird verwendet, um den Compiler davon zu überzeugen, dass eine Variable verwendet wird.

4.15.2 Dokumentation der benutzerdefinierten Typen

4.15.2.1 barrier_t

```
typedef struct barrier_t barrier_t
```

4.15.2.2 IParams

```
typedef struct IParams IParams
```

4.15.2.3 message_slot

```
typedef struct message_slot message_slot
```

4.15.2.4 process_info

```
shared_memory::process_info
```

Info zu Prozess 0. Speicher für weitere Prozess-Infos muss über die fixe Struct-Größe hinaus dynamisch berechnet werden.

4.15.2.5 shared_memory

```
typedef struct shared_memory shared_memory
```

4.15.2.6 thread_node

```
typedef struct thread_node thread_node
```

4.15.3 Dokumentation der Funktionen

4.15.3.1 calculate_shared_memory_size()

```
int calculate_shared_memory_size (
            int processes )
```

Berechnet den für den Shared Memory benötigten Speicherplatz in Abhängigkeit von der Anzahl der Prozesse.

Parameter

processes	Die Anzahl der Executable-Prozesse, die verwaltet werden.

Rückgabe

Die Größe des benötigten Speicherplatzes in Bytes.

4.15.3.2 get_process_info()

```
process_info* get_process_info (
            int rank )
```

Gibt einen Zeiger auf das process_info-Struct des angegebenen Prozesses zurück.

Parameter

rank	Rang des Prozesses, dessen process_info angefordert wird.

Rückgabe

Zeiger auf process_info-Struct des Prozesses mit dem angegebenen Rang.

4.15.3.3 OSMP_Init_Runner()

```
void OSMP_Init_Runner (
            int fd,
            shared_memory * shm,
            int size )
```

Setzt die globalen Variablen der OSMP-Bibliothek für den Elternprozess.

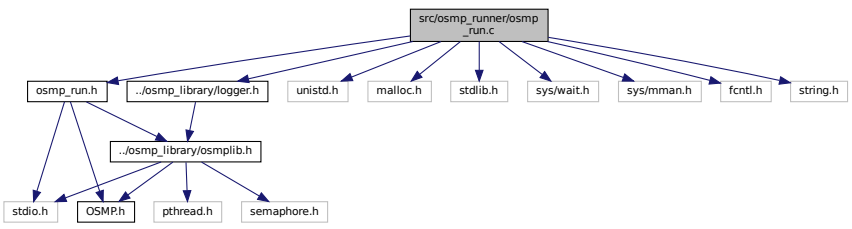
Parameter

fd	Shared-Memory-File-Descriptor.
shm	Zeiger auf den Beginn des Shared Memory.
size	Größe des Shared Memory in Bytes.

4.16 src/osmp_runner/osmp_run.c-Dateireferenz

```
#include "osmp_run.h"
#include <unistd.h>
#include <malloc.h>
#include <stdlib.h>
#include <sys/wait.h>
#include <sys/mman.h>
#include <fcntl.h>
#include <string.h>
#include "../osmp_library/logger.h"
```

Include-Abhängigkeitsdiagramm für osmp_run.c:



Funktionen

- int init_shared_mutex (pthread_mutex_t *mutex_pointer)
- int init_shared_cond_var (pthread_cond_t *cond_pointer)
- int free_all (int shm_fd, shared_memory *shm_ptr)
- void kill_threads (int count, int shared_memory_fd, shared_memory *shm_ptr)
- int start_all_executables (int number_of_executables, char *executable, char **arguments, shared_memory *shm_ptr, int shared_memory_fd)
- int is_whitespace (const char *string)
- void printUsage (void)
- void print_logfile_condition (void)
- void parse_args (int argc, char *argv[ ], int *processes, char **log_file, int *verbosity, char **executable, int *exec_args_index)
- void set_shm_name (void)
- int barrier_init (barrier_t *barrier, int count)
- int barrier_destroy (barrier_t *barrier)
- void log_pb_util_init_error (const char *format_str, int process_rank)
- void init_shm (shared_memory *shm_ptr, int processes, int verbosity)
- int destroy_postbox_utilities (postbox_utilities *postbox)
- int cleanup_shm (shared_memory *shm_ptr)
- int main (int argc, char **argv)

Variablen

- int shm_size
- char * shared_memory_name

4.16.1 Dokumentation der Funktionen

4.16.1.1 barrier_destroy()

```
int barrier_destroy (
            barrier_t * barrier )
```

Zerstört die Synchronisierungselemente (Mutex und Condition-Variable) einer Barrier.

Parameter

barrier	Zeiger auf die Barrier, deren Mutex und Condition-Variable zerstört werden sollen.

Rückgabe

OSMP_SUCCESS im Erfolgsfall, sonst OSMP_FAILURE.

4.16.1.2 barrier_init()

```
int barrier_init (
            barrier_t * barrier,
            int count )
```

Initialisiert barrier mit der angegebenen Größe count und allen Standardwerten.

Parameter

barrier	Zeiger auf die Barrier, die initialisiert werden soll.
count	Anzahl der Prozesse, die an der Barriere warten können/müssen.

Rückgabe

OSMP_SUCCESS im Erfolgsfall, sonst OSMP_FAILURE.

4.16.1.3 cleanup_shm()

```
int cleanup_shm (
            shared_memory * shm_ptr )
```

Zerstört alle Mutexe, Semaphoren und Condition-Variablen in einem Shared Memory.

Parameter

shm_ptr	Zeiger auf den Shared Memory.

Rückgabe

OSMP_SUCCESS im Erfolgsfall, sonst OSMP_FAILURE.

4.16.1.4 destroy_postbox_utilities()

```
int destroy_postbox_utilities (
            postbox_utilities * postbox )
```

Zerstört die Mutexe und Semaphoren eines postbox_utilities-Structs.

Parameter

postbox	Zeiger auf die postbox_utilities, deren Mutexe und Semaphoren zerstört werden sollen.

Rückgabe

OSMP_SUCCESS im Erfolgsfall, sonst OSMP_FAILURE.

4.16.1.5 free_all()

```
int free_all (
            int shm_fd,
            shared_memory * shm_ptr )
```

Schließt den Shared Memory (Unmapping des SHM, Schließen des FDs, Unlinken des SHM, manuell allozierten Speicherplatz für SHM-Namen freigeben).

Parameter

shm_fd	File Descriptor des Shared Memorys.
shm_ptr	Zeiger auf den Shared Memory.

Rückgabe

OSMP_SUCCESS im Erfolgsfall, sonst OSMP_FAILURE.

4.16.1.6 init_shared_cond_var()

```
int init_shared_cond_var (
            pthread_cond_t * cond_pointer )
```

Erzeugt eine Condition-Variable mit dem Attribut "shared" und kopiert sie an den gewünschten Speicherbereich.

Parameter

cond_pointer	Zeiger auf den Speicherbereich, in den die neu erzeugte Condition-Variable kopiert werden soll.

Rückgabe

OSMP_SUCCESS im Erfolgsfall, sonst OSMP_FAILURE.

4.16.1.7 init_shared_mutex()

```
int init_shared_mutex (
            pthread_mutex_t * mutex_pointer )
```

Erzeugt einen Mutex mit dem Attribut "shared" und kopiert ihn an den gewünschten Speicherbereich.

Parameter

mutex_pointer	Zeiger auf den Speicherbereich, in den der neu erzeugte Mutex kopiert werden soll.

Rückgabe

OSMP_SUCCESS im Erfolgsfall, sonst OSMP_FAILURE.

4.16.1.8 init_shm()

```
void init_shm (
            shared_memory * shm_ptr,
            int processes,
            int verbosity )
```

Setzt alle initialen Werte im fixen Teil des Shared Memory, außer folgende:

- Der Logging-Mutex wird durch den Logger gesetzt.
- Die PIDs werden in start_all_executables() gesetzt.

Parameter

shm_ptr	Pointer auf den Shared Memory.
processes	Anzahl der Prozesse.
verbosity	Logging-Verbosität.

4.16.1.9 is_whitespace()

```
int is_whitespace (
            const char * string )
```

Überprüft, ob ein String nur Leerzeichen enthält.

Parameter

string	Zeiger auf den zu überprüfenden String (muss null-terminiert sein).

Rückgabe

1, wenn der String nur Leerzeichen enthält; sonst 0.

4.16.1.10 kill_threads()

```
void kill_threads (
            int count,
            int shared_memory_fd,
            shared_memory * shm_ptr )
```

Eine Methode, die alle threads schließt.

Parameter

count	Anzahl der Threads
shared_memory↩ _fd	shared memory Dateizeiger
shm_ptr	shared memory Zeiger

4.16.1.11 log_pb_util_init_error()

```
void log_pb_util_init_error (
            const char * format_str,
            int process_rank )
```

Hilfsmethode, um formatierte Strings als Fehlermeldungen bei der Initialisierung von postbox_utiliites zu loggen.

Parameter

format_str	Formatierungsstring für die Fehlermeldung (muss genau ein d als Formatierungsanweisung enthalten).
process_rank	Rang des Prozesses, bei dessen Initialisierung ein Fehler auftritt (wird für d eingesetzt).

4.16.1.12 main()

```
int main (
            int argc,
            char ** argv )
```

4.16.1.13 parse_args()

```
void parse_args (
            int argc,
            char * argv[],
            int * processes,
            char ** log_file,
```

```
            int * verbosity,
            char ** executable,
            int * exec_args_index )
```

Diese Funktion analysiert und parst die Befehlszeilenargumente. Wenn die Argumente nicht dem geforderten Schema ./osmp_run <ProcAnzahl> [-L <PfadZurLogDatei> [-V <LogVerbosität>]] ./<osmp_executable> [<param1> <param2> ...] entsprechen, wird printUsage() aufgerufen und das Programm mit EXIT_FAILURE beendet. Achtung: exec_args_index kann == argc sein, nämlich dann, wenn keine Argumente für die OSMP-↩ Executable übergeben werden. Dies muss von der aufrufenden Funktion abgefangen werden.#define SHARED_↩ MEMORY_NAME "/shared_memory"

Parameter

in	*argc*	Die Anzahl der gesamten Kommandozeilenargumente, die an dieses Programm übergeben wurden.
in	*argv*	Zeiger auf die gesamten Kommandozeilenargumente, die an dieses Programm übergeben wurden.
out	*processes*	Zeiger auf die Anzahl der Prozesse, die gestartet werden sollen.
out	*log_file*	Zeiger auf den Namen des Logfiles. Wird auf NULL gesetzt, wenn argv keine Logdatei angibt.
out	*verbosity*	Zeiger auf die Log-Verbosität. Wird auf 1 gesetzt, wenn argv keinen oder einen ungültigen Wert enthält.
out	*executable*	Zeiger auf den Namen der Executable. Wird auf NULL gesetzt, wenn in den Argumenten nicht gesetzt oder leer.
out	*exec_args_index*	Zeiger auf den Index in Bezug auf argv, an dem das erste an die OSMP-Executable zu übergebende Argument steht (den Namen der Executable nicht eingeschlossen).

4.16.1.14 print_logfile_condition()

```
void print_logfile_condition (
            void  )
```

Gibt einen Hinweis zur maximalen Pfadlänge der Logdatei aus.

4.16.1.15 printUsage()

```
void printUsage (
            void  )
```

Gibt die korrekte Verwendung des Programms aus.

4.16.1.16 set_shm_name()

```
void set_shm_name (
            void  )
```

4.16.1.17 start_all_executables()

```
int start_all_executables (
            int number_of_executables,
            char * executable,
            char ** arguments,
            shared_memory * shm_ptr,
            int shared_memory_fd )
```

4.16.2 Variablen-Dokumentation

4.16.2.1 shared_memory_name

```
char* shared_memory_name
```

Shared memory name to be created following the scheme: shared_memory_<runner_pid>

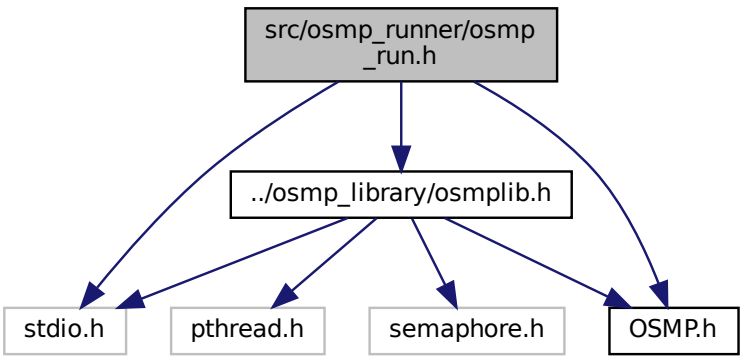
4.16.2.2 shm_size

```
int shm_size
```

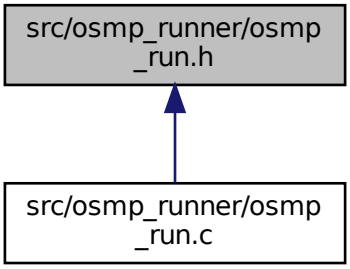
4.17 src/osmp_runner/osmp_run.h-Dateireferenz

```
#include <stdio.h>
#include "../osmp_library/osmplib.h"
#include "../osmp_library/OSMP.h"
```

Include-Abhängigkeitsdiagramm für osmp_run.h:



Dieser Graph zeigt, welche Datei direkt oder indirekt diese Datei enthält:



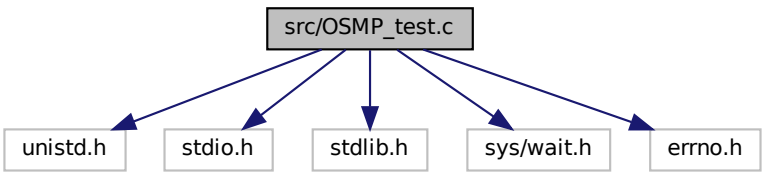
Klassen

- struct monitor_args

4.18 src/OSMP_test.c-Dateireferenz

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/wait.h>
#include <errno.h>
```

Include-Abhängigkeitsdiagramm für OSMP_test.c:



Funktionen

- int main (void)

4.18.1 Dokumentation der Funktionen

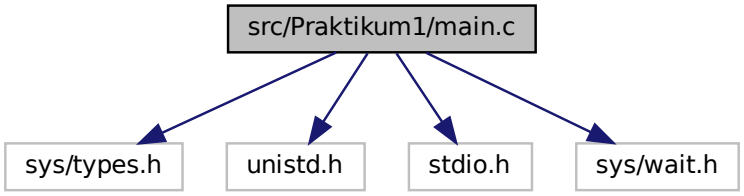
4.18.1.1 main()

```
int main (
            void  )
```

4.19 src/Praktikum1/main.c-Dateireferenz

```
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
#include <sys/wait.h>
```

Include-Abhängigkeitsdiagramm für main.c:



Funktionen

- int main (void)

4.19.1 Dokumentation der Funktionen

4.19.1.1 main()

```
int main (
            void  )
```

Index