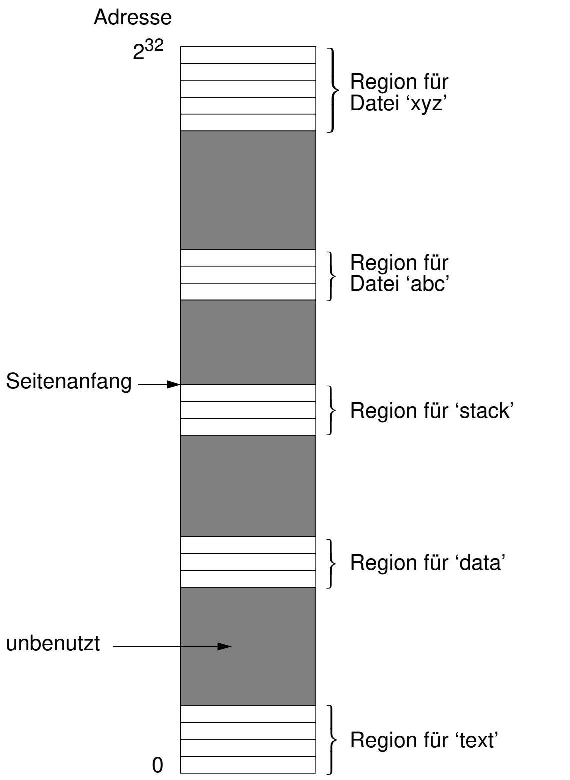
5.3 Arbeitsspeicherverwaltung

Arbeitsspeicherverwaltung in Mach

- ❏ flexible Verwaltung basierend auf Seitenadressierung
- ❏ Trennung in maschinenabhängigen und maschinenunabhängigen Teil auf eine sehr innovative Weise
- ❏ Drei Module mit klar definierten Schnittstellen:
  - Das erste Modul läuft im privilegierten Modus und verwaltet die MMU, ist daher hochgradig maschinenabhängig.
  - Das zweite Modul läuft ebenfalls im privilegierten Modus, ist aber maschinenunabhängig. Es behandelt Seitenfehler, Adreßräume und Seitenersetzung.
  - Das dritte Modul läuft als Benutzerprozeß. Es behandelt hauptsächlich den logischen Teil der Speicherverwaltung insbesondere die Hintergrundspeicher (memory manager, external pager).
  - Kommunikation zwischen Modulen durch Nachrichten an ‘ports’, die an Module gebunden sind
- ❏ Virtueller Speicher
  - Adreßräume sind linear
  - Effiziente Handhabung dünn besetzter Adreßräume durch das Konzept der Regionen (regions)
  - Regionen können zugeteilt und freigegeben werden, Anfangsadresse ist bei Zuteilung angebbar.



- Das ‘virtual memory interface’ besteht aus folgenden Methoden (Benutzerfäden an Kern)

Methode	Beschreibung
vm_allocate target_task address size anywhere	Eine Region virtueller Adressen zugänglich machen
vm_deallocate target_task address sizw	Eine Region von Adressen unzugänglich machen
vm_map target_task address size mask anywhere memory_object offset copy cur_protection max_protection inheritance	Ein Speicherobjekt in den virtuellen Adreß-raum abbilden
vm_copy target_task source_address count dest_address	Kopie einer Region an einer anderen Stelle des virtuellen Adreßraums anlegen
vm_inherit target_task address sixe new_inheritance	Vererbungsattribut für eine Region setzen

Methode	Beschreibung
vm_read target_task address size data data_count	Einen Bereich aus dem virtuellen Adreß-raum eines anderen Prozesses lesen
vm_write target_task address data data_count	Einen Bereich in den virtuellen Adreßraum eines anderen Prozesses kopieren

•

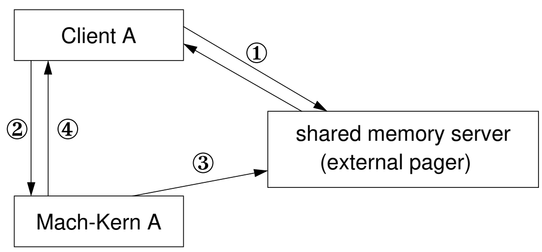
• Nachrichten des Kerns an den ‘external pager’ (memory_object_)

Nachricht	Beschreibung
init	Initialisieren eines neu abgebildeten Speicherobjektes
data_request	Dem Kern ist die wegen eines Seitenfehlers benötigte Seite zu übergeben
data_write	Rückgabe einer modifizierten Seite
data_unlock	Anforderung von Zugriffsattributen durch den Kern
lock_completed	Eine frühere Aufforderung zum Rückschreiben modifizierter Seiten ist vollzogen
terminate	Anzeige, daß das Objekt zurückgegeben ist und nicht mehr benutzt wird

Nachrichten des ‘external pager’ an den Kern (memory_object_)

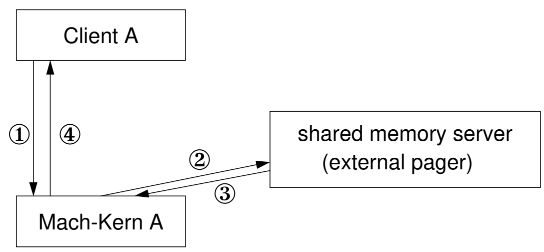
Nachricht	Beschreibung
set_attributes	Übergabe von Attributen an den Kern z. B. zur Steuerung der Ersetzungsstrategie
data_provided	Übergabe der angeforderten Seite (Antwort auf data_request)
data_unavailable	Angeforderte Seite ist nicht verfügbar (Antwort auf data_request)
lock_request	Aufforderung an den Kern modifizierte Seiten zurückzuschreiben, zu invalidieren oder Zugriffsattribute zu ändern (wird beantwortet mit lock_completed)
destroy	Aufforderung an den Kern, das Objekt zurückzugeben (wird beantwortet mit terminate)

Beispiel: Initialisierung



① get_object
② allocate + map
③ init
④ client resumed

Seitenfehler



① Zugriff mit Seitenfehler
② data_request
③ data_provided
④ client resumed

Verteilter, gemeinsam genutzter Speicher

❑ Anforderungen
  - Beispiel: parallele Bearbeitung von Satellitendaten mit unterschiedlichen Programmen
  - Differenzierte Behandlung der Regionen bei fork: ‘text’-Segmente können gemeinsam genutzt werden
  - manche Regionen (z. B. solche die Dateien beherbergen) können für den erzeugten Prozeß irrelevant sein
  - bei manchen Regionen kann der Anfangszustand relevant sein, aber es sollen erzeugender und erzeugter Prozeß eigene Kopien besitzen

❑ Antwort in Mach: Vererbungsattribute
  - Region wird vom erzeugten Prozeß nicht benötigt
  - Region wird von erzeugendem und erzeugtem Prozeß kooperativ benutzt
  - erzeugter Prozeß erhält Kopie der Region

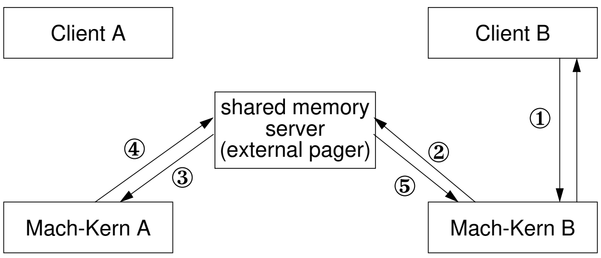
❑ Organisation
  - externer Speicherverwalter, läuft auf Benutzerebene
  - drei ports
    - ‘object port’ beim Speicherverwalter
    - ‘control port’ im Kern als Adressat für Nachrichten eines Speicherverwalters
    - ‘name port’ wird als eine Art Name für das Speicherobjekt benutzt (z. B. kann zu einer vorgebbaren virtuellen Adresse erfragt werden, zu welchem Speicherobjekt sie gehört. Die Antwort besteht aus einem Zeiger auf den ‘name port’.)

❑ Gemeinsam genutzter Speicher

Sowohl für NORMA- als auch für NUMA-Architekturen von Interesse

Wichtige Gesichtspunkte
  - Struktur und Granularität
  - Kohärenz-Semantik
  - Skalierbarkeit
  - Implementierung
    - Datenlokalisierung und Zugriff
    - Kohärenzprotokoll (write-invalidate, write-update)
    - Ersetzungsstrategie
    - Thrashing
    - Interaktionsmechanismen

Gemeinsam genutzter Speicher,von A angesprochene Seite wird von B lesend angesprochen



① Zugriff mit Seitenfehler
② data_request
③ lock_request (read only)
④ lock_completed
⑤ data_provided (read only)

Speicherkohärenz

Strikte Konsistenz
Lesen liefert zuletzt geschriebenen Wert

↓

Sequentielle Konsistenz
Die Ausführung erscheint als zeitliche Durchmischung der Operationen, wie sie bei Mehrprozeßbetrieb auftreten kann.

↓

Prozessorkonsistenz
Schreibaufrufe eines Prozessors werden immer in der Reihenfolge ihrer Anfordrung sichtbar. Über Anforderungen von verschiedenen Prozessoren kann keine Aussage über die Reihenfolge gemacht werden, in der ihre Ausführung sichtbar wird.

Schwache Konsistenz
Koordinierungsoperationen sind sequentiell konsistent.
Vor Zugriff auf Koord.-Variablen müssen alle Schreibvorgänge abgeschlossen sein.
Zugriff auf Daten nur, wenn alle vorherigen Zugriffe auf Koord.-Variablen abgeschlossen sind.

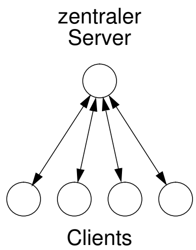
↓

Freigabe (Release) Konsistenz
Jede der beiden Koordinierungsoperationen acquire und release ist prozessorkonsistent
Vor Ausführung von release müssen alle vorherigen Schreib- und Lesevorgänge des Prozesses abgeschlossen sein

• Algorithmen für die Plazierung von Seiten

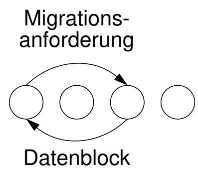
	nicht vervielfachend	vervielfachend
Nicht migrierend	zentralisiert (central server)	allgemein verviel-fachend (full-replication)
migrierend	migrierend (migration)	vervielfachend für Lesezugriff (read-replication)

- Zentralisierter Server (Central-server algorithm)



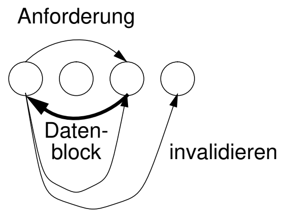
Client	zentraler Server
Datenanforderung senden	
	Anforderung entgegennehmen Datenzugriff durchführen Antwort senden
Antwort entgegennehmen	

- Migrationsalgorithmus



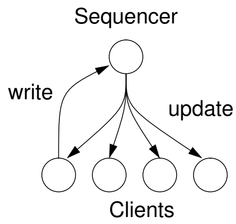
Client	Verwalter
Falls Block nicht lokal, Speicherort ermitteln und Anforderung senden	
	Anforderung entgegennehmen Block senden
Antwort entgegennehmen Datenzugriff durchführen	

- Vervielfachung für lesenden Zugriff



Client	Verwalter
Falls Block nicht lokal, Speicherort ermitteln und Anforderung senden	
	Anforderung entgegennehmen Block senden
Antwort entgegennehmen Invalidierung an alle anderen	
	Invalidierung vollziehen
Zugriff	

- Allgemein vervielfachend



Client	Sequencer	Verwalter
Falls 'write' Daten an Sequencer		
	Sequenznummer anfügen	
	Multicast	
		Daten empfangen 'update' lokal
Ausführung bestätigt		
'update' lokal		

- Vergleichende Analyse

  Stumm, M.; Zhou, S.: Algorithms Implementing Distributed Shared Memory. Computer, May 1990, pp. 54-64.