Kapitel 7: Kommunikation

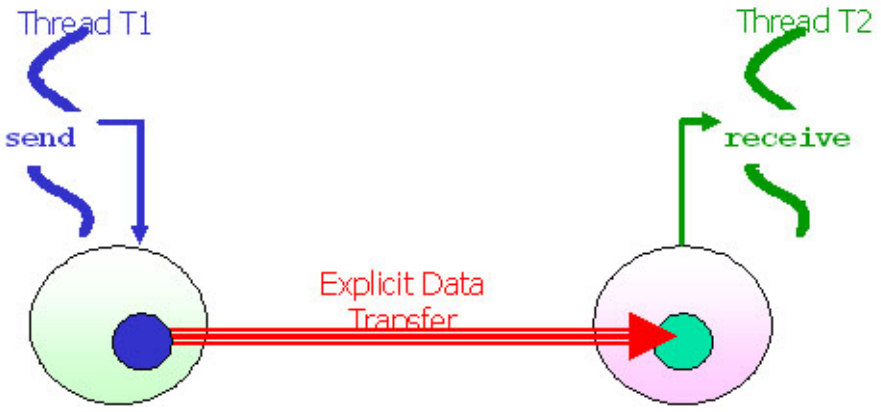
- Motivation und Einführung
- Kommunikationsmodell
- Elementare Kommunikation
  - Orthogonale Entwurfsparameter
  - Synchronisation
  - Adressierung
  - Datentransfer
- Höhere Kommunikation
- Kommunikationsbeispiele

7.1 Motivation und Einführung

Spätestens in verteilten Systemen wird dieser Interaktionsmechanismus zusätzlich benötigt, manchmal ist es aber auch in lokalen Systemen vorteilhaft, dass man trotz vorhandenem gemeinsamen Hauptspeicher aus Schutzgründen Kommunikation zwischen Prozessen oder Tasks einrichtet, wenn sie sich wechselseitig nicht so ganz trauen. Bei einer Kommunikation findet ein Nachrichtenaustausch statt, d.h. eine gewisse Informationsmenge (Nachricht) wird vom Sender(adressraum) zu einem Empfänger(adressraum) transferiert. Wie im täglichen Leben auch mag es Kommunikationsanwendungen geben, in denen nur wenig Information fließen muss (siehe militärische Kommandos) und solchen, in denen ein Fülle von Daten im Spiel sind (siehe meteorologische Daten).

7.2 Kommunikationsmodell

Bei einer Kommunikation sind mindestens eine sendende Instanz und eine empfangende Instanz beteiligt. Obwohl es sich bei den beteiligten Kommunikationsinstanzen um jede der möglichen Aktivitätsformen (Prozess, KLT oder PULT) handeln kann, wollen wir uns im Folgenden wiederum nur auf KLT fokussieren, wobei im Regelfall zwei KLTs verschiedener Adressräume miteinander kommunizieren sollen.



Im Kommenden müssen wir also berücksichtigen, wie Information gezielt aus einem Adressraum in einen anderen transferiert werden kann, wann dieser Informationsfluss stattfinden soll und worauf sich die beteiligten Kommunikationspartner verständigen können. Da es sich bei den Kommunikationspartnern um KLTs handelt, werden wir demzufolge primär IPC-Objekte studieren, die an der Kernschnittselle angeboten werden, demzufolge auch nur über Systemaufrufe genutzt werden können.

7.3 Elementare Kommunikation

Zum besseren Verständnis werden wir zunächst vereinfachend annehmen, dass genau ein Sender- und genau ein Empfänger-KLT an der zu untersuchenden unidirektionalen Kommunikation beteiligt sind. Wie sieht

unter diesen vereinfachten Randbedingungen die Spielwiese der Systemarchitekten aus, oder in anderen Worten, welche orthogonalen Entwurfsparameter können bei der elementaren Kommunikation unterschieden werden?

- Verbindungsart der Kommunikationspartner
- Synchronisation der Kommunikationspartner
- Adressierung der Kommunikationspartner
- Zugehörigkeit bzw. Besitztum von Kommunikationsobjekten
- Organisation des Nachrichtentransports

In den folgenden Abschnitten werden diese Gestaltungsparameter systematisch, wenngleich nicht erschöpfend behandelt.

7.3.1 Verbindungsart der Kommunikationspartner

Man kann (insbesondere in verteilten Systemen) zwischen verbindungsorientierter und verbindungsloser Kommunikation unterscheiden. Man wird i.d.R. die erstere Variante wählen, wenn die Kommunikationspartner über einen längeren Zeitraum immer wieder Nachrichten über genau diese Verbindung und keine andere austauschen möchten.

Beim Aufbau der Kommunikationsverbindung (z.B. mittels `openConnection(address)` wird man aus der Sicht des Senders den Zieladressaten angeben, so dass dann bei den nachfolgenden Sende- bzw. Empfangsoperationen über diese Kommunikationsleitung keine Adressen beigefügt werden müssen. Allerdings muss es aus der Sicht des Senders und Empfängers eindeutig sein, dass nur eine Verbindung gemeint sein kann.

Im Gegensatz dazu wird es bei einer verbindungslosen Kommunikation bei jeder Sende- bzw. Empfangsoperation nötig sein, die jeweilige Verbindung (z.B. Kommunikationskanal) bzw. die Adresse des Kommunikationspartners explizit in den Kommunikationsoperationen zu nennen.

7.3.2 Synchronisation der Kommunikationspartner

Wie im täglichen Leben gibt es auch in Systemen verschiedene Möglichkeiten, wie sich die Kommunikationspartner wechselseitig informieren können.

Beispielsweise ist es ja bei Geheimdiensten üblich, sich so genannter *"toter Briefkästen"* zu bedienen. Ein Agent, der seine ausspionierte Nachricht über einen Mittelsmann an seine Behörde weitergeben möchte, legt diese Information in diesem "toten Briefkasten" *asynchron* ab. Der Mittelsmann wiederum überprüft -um möglichst wenig aufzufallen- in unregelmäßigen Abständen diesen toten Briefkasten, ob mittlerweile wieder eine Nachricht abgelegt worden ist. Ganz andere *Zwänge* können bei folgendem Beispiel festgestellt werden; zwei frisch Verliebte werden wohl nur ein Rendezvous als einzig sinnvolle Form der zwischenmenschlichen Kommunikation empfinden.

In Systemen werden mitunter alle möglichen Varianten der synchronen bzw. asynchronen Kommunikation angeboten, um so Anwenderprogrammierer möglichst wirkungsvoll in verschiedenen Anwendungsfällen zu unterstützen. Die folgenden Kombinationen aus nicht blockierenden Sende- bzw. Empfangsoperationen haben sich bei klassischen monolithischen Kernen durchgesetzt:

	Non-blocking Receive	Blocking Receive
Unsynchronized Send	- bogus -	Sender polling
Asynchronous Send	Receiver polling	Synchronous communication
Synchronous Send	Receiver polling	Rendevous (ADA)

Andererseits ist es speziell in Mikrokernen ein Grund mehr, sich darüber Gedanken zu machen, welche dieser Kombinationen implementiert werden soll, will man den Grundgedanken des Mikrokerns nicht *ad absurdum* führen. Jochen Liedtke hat sich beim Entwurf von L4 dazu entschieden, nur synchrone IPC anzubieten.

Verständnisfragen:

1. *Warum wurden nicht alle Kombinationen angeboten?*
2. *Warum wird ausgerechnet das Rendezvous angeboten?*

Hinweis: In Systemen, die auch asynchrones Senden anbieten, muss im Kern oder an anderer Stelle Pufferplatz für die Nachrichten angeboten werden, die nicht sofort zum Empfänger weitergeleitet werden können.

Spätestens in verteilten Systemen muss man damit rechnen, dass Nachrichten entweder wegen Netzwerkfehler ihr Ziel nicht erreichen oder dass vom Empfänger keine Eingangsbestätigung zurückkommt, so dass man auf Senderseite im Ungewissen bleibt, ob die Kommunikation nun geklappt hat oder nicht. Deshalb möchte man insgesamt die Nachrichtenübermittlungszeit gerne individueller kontrollieren können.

Hierzu dient der Zeitauslaufmechanismus (*time out*), mit dessen Hilfe ein Kommunikationspartner ein mögliches Fehlverhalten seines Kommunikationspartners oder einen Fehler in der Nachrichtenübermittlung vermuten kann.

7.3.3 Adressierung des Kommunikationspartners

Es gibt prinzipiell zwei verschiedene Adressierungsarten, wie man eine Nachricht auch tatsächlich zum gewünschten Kommunikationspartner übermitteln kann.

7.3.3.1 Direkte Kommunikation

Man kennzeichnet die Zieladresse innerhalb der Sendeoperation, indem man entweder

- den "Namen" des Kommunikationspartners oder dessen
- systemweit eindeutige ID oder
- dessen TCB Adresse

angibt.

Im ersten Fall muss hierzu ein Namensdienst im System eingerichtet sein, der aus dem Namen dann eine ID ermittelt, die dann den eigentlichen Parameter der Sendeoperation darstellt. Viele direkten Kommunikationen werden über TIDs abgewickelt, deren Abbildung auf TCBs i.d.R. über Hashtabellen erfolgen kann.

Die direkte Verwendung von TCB-Adressen ist eh nur im Kern. Deren wesentlicher Nachteil besteht darin, dass bei einer Migration eines Dienstes oder bei der Ersetzung eines alten Dienstes durch einen verbesserten Dienst die Programme neu übersetzt werden müssten, da sich i.d.R. die Adressen der beteiligten TCBs verändern werden.

Ein prinzipieller Vorteil der direkten Adressierung besteht darin, dass man diese Kommunikation effizient implementieren kann, insbesondere wenn man sich darauf beschränkt, nur synchrone:synchrone Kommunikation zuzulassen (siehe hiezu den Mikrokern L4).

Nachteilig mag sich auswirken, dass man z.B. bei einem dynamisch sich selbst replizierenden multi-threaded Server diesen nicht direkt adressieren kann, sondern dass man stets hierzu einen Verteiler-Thread (*dispatcher thread*) benötigt, der die Aufträge der Klienten entgegennimmt, um diese dann nach einem Schema an den nächsten Arbeiterthread (*worker thread*) weiterzuleiten. In einer zentralen Serverdatenstruktur können sich neu replizierte Arbeiter an- und abmelden.

7.3.3.2 Indirekte Kommunikation

Bei einer indirekten Kommunikation erfolgt die Ablage einer Nachricht in ein Kommunikationsobjekt, z.B. in einen Briefkasten (*mailbox*), Nachrichtenkanal (*channel or message queue*) oder in einem Port. Auf diese Weise kann man beliebige Kommunikationsmuster vom Typ m:n-Verbindungen etwas leichter als mittels direkter Adressierung aufbauen. Ferner kann man hiermit persistente Kommunikationseinrichtungen etablieren, die insbesondere die Lebensdauer ihrer Kommunikationsteilnehmer überleben können.

7.3.4 Besitztum der Kommunikationsobjekte

Ein Port wird i.d.R. zusammen mit einem Thread erzeugt, der als Besitzer dieses Ports denselben als Posteingangsstelle verwendet. Mailboxes können unabhängig von ihrem Erzeuger und den mit ihnen kommunizierenden Partner existieren, also insbesondere deren Existenz auch überleben.

7.3.5 Organisation des Nachrichtentransports

Es gibt prinzipiell verschiedene Entwurfs- und Implementierungsvarianten, wie man Nachrichten aus dem Senderpuffer in den Zielpuffer beim Empfänger transferieren kann.

Die Nachrichtenlänge und der Zusammenhang einer Nachricht sind zwei Gestaltungsparameter, die den Entwurf des Datentransfers signifikant beeinflussen können, wir wollen deshalb einen Unterschied zwischen folgenden Nachrichtentypen machen und hierzu jeweils mindestens eine Implementierungsvariante angeben:

Kurze zusammenhängende Nachricht	(in Registern)
Lange zusammenhängende Nachricht	(per copy-in-copy-out, per mapping)
Nichtzusammenhängende Nachrichtenblöcke	(Anfangsadressen in (virtuellen) Registern)

Man beachte allerdings, dass obige Form des Datentransfers nicht ganz unabhängig von der Synchronisation der Kommunikation realisiert werden kann.

Beispielsweise kann unkontrolliertes asynchrones Senden von kurzen Nachrichten dazu führen, dass Kurznachrichten im Register schlichtweg überschrieben werden und somit verloren gehen.

Systeme, die an der Kern-API Kommunikationsoperationen anbieten, verzichten manchmal darauf, noch extra Synchronisationsoperationen dort anzubieten, da man auch mit geeigneten Kommunikations-operationen viele Synchronisationsaufgaben lösen kann.

7.4 Höhere Kommunikation

In modernen Systemen, die dem Komponenten orientierten Ansatz folgen, werden auch lokal einige Systemdienste in Form von Prozessen oder Tasks außerhalb des Kerns angeboten werden. Aus dem Umfeld der verteilten Systeme verwendet man quasi als Standard den entfernten Prozeduraufruf (RPC), um aus der Sicht eines Klienten einen Auftrag oder eine Anfrage an einen Server zu stellen. Man kann diesen Ansatz auch in lokalen Systemen übernehmen, um zwischen Anwendungen und Dienstgebern (server) Serveraufträge bzw. deren Ergebnisse miteinander auszutauschen.

Problem: Warum wird man sich z.B. sogar in einem Mikrokern à la L4 den Luxus erlauben, an der Mikrokernschnittstelle einen Call-Mechanismus zusätzlich anzubieten, wenn man doch denselben Zweck mit einem Senden_Auftrag und unmittelbar anschließendem Empfangen_Ergebnis auch ausrichten könnte?

Zum einen benötigt man nur noch einen einzigen Systemaufruf, man spart auf Klientenseite also schon mal zig-Zyklen (je nach Hardware). Gravierender schlägt jedoch zu Buche, dass hierdurch ein unbeabsichtigter Seiteneffekt eintreten kann.

Nehmen wir an, dass zwischen Sendeaufruf und dem direkt darauf folgenden Empfangsaufruf im Klienten der Zeitscheibenmechanismus zuschlägt und der Kernscheduler dem Klienten den Prozessor entzieht. Der Server wird dann beispielsweise rechnen und mit seinem Auftrag fertig werden, dann versucht der Server vergeblich sein Ergebnis an den Klienten abzusetzen, was nicht geht, da dieser ja noch nicht die Empfangsoperation aufrufen konnte. Also wird der Kernscheduler den Server blockieren und auf einen anderen KLT umschalten, das muss aber nicht der Klient sein, dem er zuvor den Prozessor entzogen hatte, sondern es können noch sehr viel andere KLTs dazwischen kommen, von denen einige bereits wieder Aufträge an den Server haben, diese aber ebenso wenig absetzen können, da der Server ja immer noch blockiert ist, da er sein zuvor erbrachtes Ergebnis nicht weiterleiten konnte.

Manche mögen nun einwenden, dass dieser Fall konstruiert ist und in der Praxis kaum vorkommen wird. Was aber, wenn doch? Hierzu sollte man sich nicht nur die häufig zitierten Murphy's-Gesetze zur Belustigung anschauen, sondern sich eher die vielen Softwarekatastrophen vor Augen führen, an denen allzu sorglose Systemhacker ihren Anteil hatten.

Andere Beispiele von höherer Kommunikation schließen all die Anwendungen ein, in denen auf beiden Seiten mehr als ein KLT beteiligt sind, also bis zu s>1 Sender und bis zu e>1 Empfänger.

7.5 Kommunikationsbeispiele

In Unix V Rel. 4 wurden neben bis dahin gebräuchlichen Pipes (und den weniger handlichen Signalen) sowohl das Konzept des Shared Memory inklusive Semaphoroperationen, als auch ein indirektes Kommunikationsschema auf der Basis von Kommunikationskanälen eingerichtet.

Eruieren Sie, welche Kommunikationsobjekte bzw. --mechanismen in WindowsXP angeboten werden!