Interprocess Communication (IPC) in comparison to the Message Passing Interface (MPI) in a microkernel context

Janos Zweifel and Manuel Hermenau

Overview

- Message Passing Interface (MPI)
- Interprocess Communication (IPC)
- Comparison between MPI and IPC
- IPC realisable by implementing MPI?

Message Passing Interface

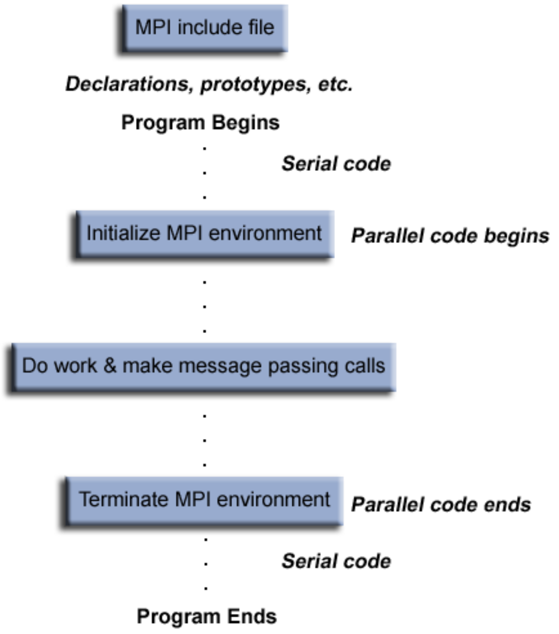
- Standard for parallelization
- Started in 1992
- First stable release in 1994
- Current version: MPI 3.0 (released 2012)

Message Passing Interface

- Originally developed for distributed memory
- Now: support for every type(distributed, shared, hybrid)
- User still sees a distributed memory system
- Key Features: standardization, portability, performance, functionality, availability

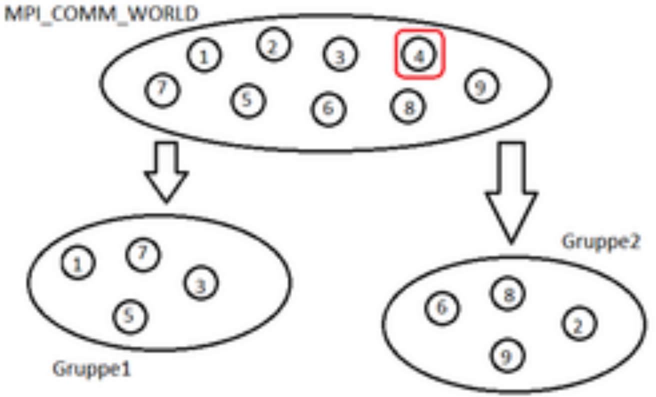
Message Passing Interface

- MPI Environment:
  - needs to be initialized
  - uses IDs (ranks)
  - uses groups and communicators (clans & chiefs)



Message Passing Interface

- Communicators and groups
  - one communicator per group
  - every process has unique ID within group
  - communication with process beyond group via communicator

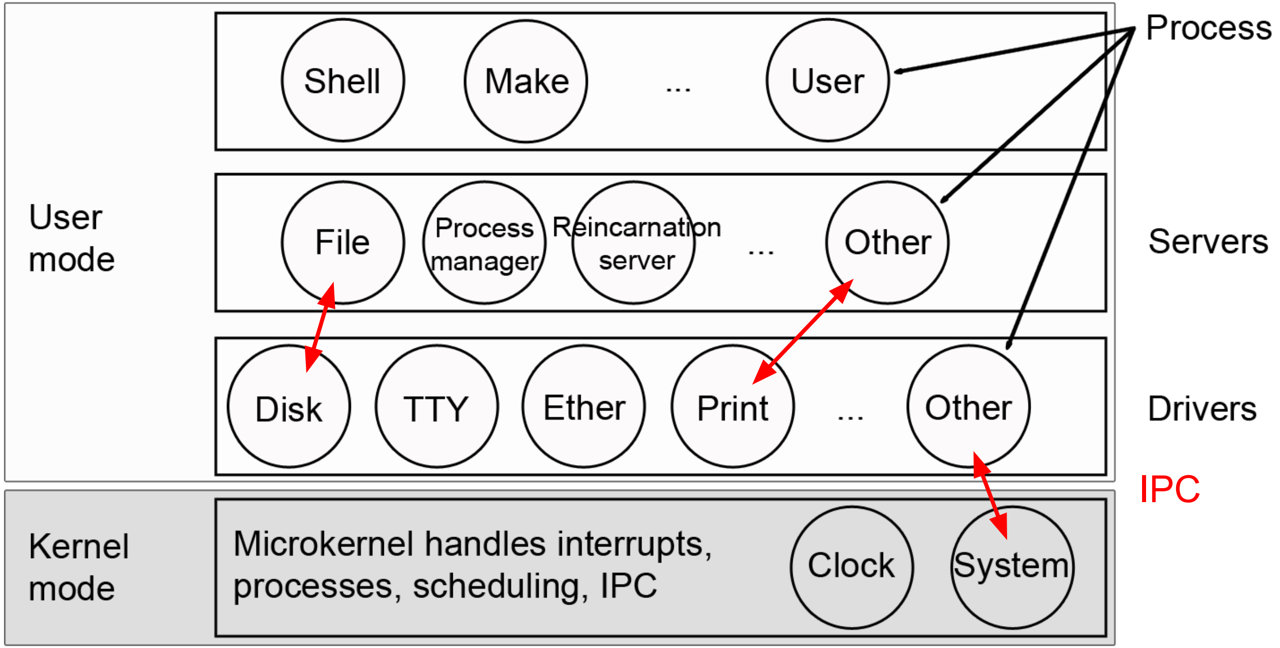


Message Passing Interface

- Common operations
  - Synchronous/Asynchronous Send/Receive
  - Blocking Send/Receive
  - Non-Blocking Send/Receive
- Often used
  - scatter
  - gather
  - broadcast

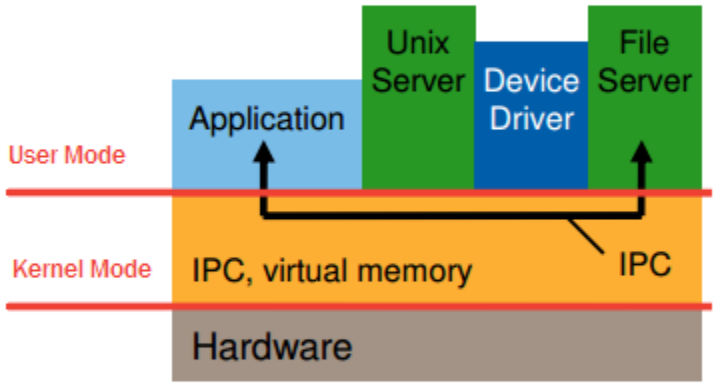
Interprocess Communication

- cross-address space communication



Interprocess Communication

- cross-address space communication



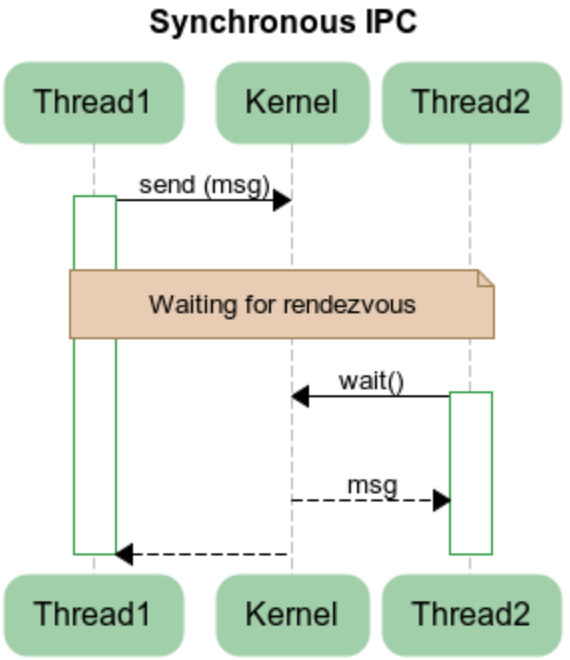
Interprocess Communication

- IPC is the glue in a microkernel system
- Performance is crucial!
- There is no “the IPC” : many different approaches

IPC Operations

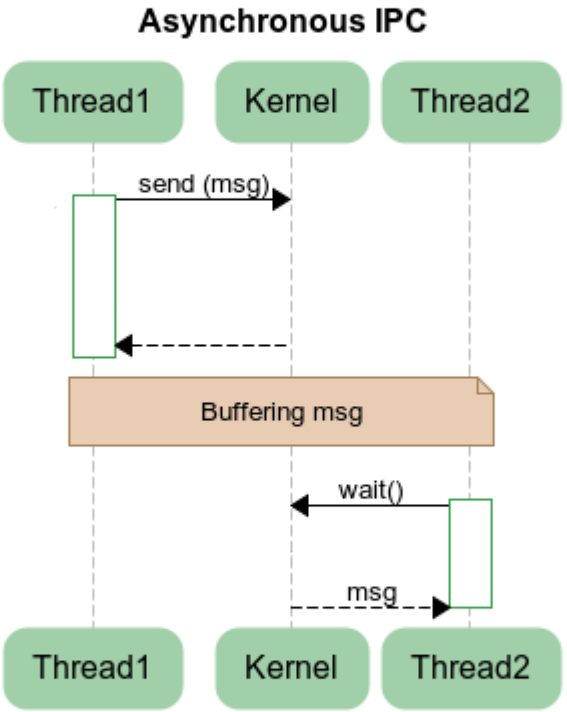
- send
  - send to specific receiver
  - reply
- receive
  - receive from specific sender
  - wait for a message from any sender
- later (to save system calls):
  - send & receive
  - reply & wait

IPC Operations



- blocks until sender and receiver are ready ("rendez-vous")

IPC Operations



- requires buffering, which can be a big overhead

IPC Communication Control

- Clans & Chiefs (L3)
  - target identified by thread id
- Capabilities (Fiasco.OC, seL4, Mach, ...)
  - Capabilities grant access to a communication channel
  - Additional objects
  - Bookkeeping required

Comparison between MPI and IPC

- MPI
  - Synchronous and asynchronous Send/Receive available
  - Communication via IDs for within the group and via communicator for beyond the group
  - Methods to transmit primitives and user-based data-types
  - Synchronization via barriers

Comparison between MPI and IPC

- IPC
  - Depending on the kernel:
    Synchronous, Asynchronous or both
  - Depending on the kernel:

    Communication via the UTCB and capabilities or via Clans & Chiefs
  - Data transfer via the UTCB (needs to be casted and/or maybe otherwise processed)

Replacing IPC with MPI

Would it be possible to model microkernel IPC after
the MPI standard and what would be the advantages and disadvantages ?

Replacing IPC with MPI

Pro:

- Standardisation

Contra:

- Commitment to concepts, like asynchronous IPC and Clans & Chiefs

Replacing IPC with MPI

Some numbers...

	L4 IPC	MPI
Specification pages	20 (total: 218)	822
LOC	~1.700 (Fiasco.OC)	830.447[1] (OpenMPI)

> “ The basic idea of the µ-kernel is [...] to implement outside of the kernel whatever possible. ”
>
> J.Liedtke, 1995

[1] http://www.ohloh.net/p/openmpi/analyses/latest/languages_summary (11.02.2014)

Replacing IPC with MPI

Pro:

- Standardisation

Contra:

- Commitment to concepts, like asynchronous IPC and Clans & Chiefs
- Unnecessary overhead, like I/O interface
- Huge code blowup

Replacing IPC with MPI

Would it be possible to model microkernel IPC after
the MPI standard and what would be the advantages and disadvantages ?

No, not without abandoning
basic microkernel concepts !

Questions?