Developing a Multiserver Operating System

Jakub Jermář
February 3, 2010
UINX.CZ
What is a Multiserver OS?
What is a Multiserver OS?

- microkernel-based OS, which is...
What is a Multiserver OS?

- microkernel-based OS, which is...
- ...multiserver
What is a Multiserver OS?

- microkernel-based OS, which is...
- ...multiserver
  - composed of multiple server tasks
What is a Multiserver OS?

- microkernel-based OS, which is...

- ...multiserver
  - composed of multiple server tasks
What is a Multiserver OS?

- microkernel-based OS, which is...
- ...multiserver
  - composed of multiple server tasks
- not every microkernel-based OS is multiserver
What is a Multiserver OS?

• microkernel-based OS, which is...
• ...multiserver
  – composed of multiple server tasks
• not every microkernel-based OS is multiserver
• not every OS is microkernel-based
OS Classification by Architecture

microkernel

server task

app task

multiserver-microkernel
OS Classification by Architecture

- Microkernel
- Server task
- Task
- System task
- App task
- Multiserver-microkernel
- Microkernel with single system task
OS Classification by Architecture

- **multiserver-microkernel**
- **microkernel**
- **microkernel with single system task**
- **monolithic kernel**

**Diagrams:**
- Microkernel
- Multiserver-microkernel
- Microkernel with single system task
- Kernel
The Multiserver Advantage

multiserver-microkernel

microkernel

microkernel with single system task

monolithic kernel
The Multiserver Advantage

- app task
- server task
- microkernel
- multiserver-microkernel

- system task
- microkernel with single system task

- kernel
- monolithic kernel
The Multiserver Advantage

- multiserver-microkernel
- microkernel with single system task
- monolithic kernel

- app task
  - app task
  - app task
- server task
  - server task
  - server task

- microkernel

- system task
The Multiserver Advantage

- multiserver-microkernel
- microkernel with single system task
- monolithic kernel
The Multiserver Advantage

- app task
- server task
- microkernel
- multiserver-microkernel

- app task
- server task
- microkernel with single system task

- app task
- server task
- monolithic kernel
The Multiserver Advantage

- microkernel
- multiserver-microkernel
- microkernel with single system task
- monolithic kernel

- app task
- server task

- app task
- server task

- app task
- server task
The Multiserver Advantage

- App task
- Server task
- Microkernel

Multiserver-microkernel

Microkernel with single system task

Monolithic kernel
Pros and Cons Overview
Pros and Cons Overview

Pros

- Improved robustness and fault isolation
- Clean interface between servers
- Simpler components
- Flexibility in connecting components
Pros and Cons Overview

**Pros**
- Improved robustness and fault isolation
- Clean interface between servers
- Simpler components
- Flexibility in connecting components

**Cons**
- Worse performance
- No cross-layer optimizations
Multiserver-Microkernel Examples
Multiserver-Microkernel Examples

Hurd
http://hurd.gnu.org
Multiserver-Microkernel Examples

Hurd
http://hurd.gnu.org

MINIX 3
http://minix3.org
Multiserver-Microkernel Examples

- Hurd
  http://hurd.gnu.org

- MINIX 3
  http://minix3.org

- HelenOS
  http://helenos.org
Simplified HelenOS Architecture
Simplified HelenOS Architecture
Simplified HelenOS Architecture

- MBR partition driver
- GUID partition driver
- file backed block device
- Clipboard service
- RAM disk driver
- DEVFS
- FAT
- TMPFS
- ATA disk driver
- Device mapper
- VFS
- Char mouse driver
- Naming service
- Task monitor
- Console server
- i8042 driver
- microkernel SPARTAN
- Framebuffer service

[Diagram with various components and services labeled]
Simplified HelenOS Architecture

- Ethernet
- nildummy
- TCP
- UDP
- dp8390 driver
- loopback driver
- ICMP
- IP
- ARP
- MBR partition driver
- GUID partition driver
- file backed block device
- Network packet server
- Clipboard service
- RAM disk driver
- DEVFS
- FAT
- TMPFS
- Keyboard Service
- ATA disk driver
- Device mapper
- VFS
- Char mouse driver
- i8042 driver
- Naming service
- Task monitor
- microkernel SPARTAN
- Console server
- Framebuffer service
Simplified HelenOS Architecture

- Ethernet
- nildummy
- Application
- TCP
- UDP
- dp8390 driver
- loopback driver
- ICMP
- IP
- ARP
- MBR partition driver
- GUID partition driver
- file backed block device
- Network packet server
- Clipboard service
- RAM disk driver
- DEVFS
- FAT
- TMPFS
- Keyboard Service
- ATA disk driver
- Device mapper
- VFS
- Char mouse driver
- i8042 driver
- Naming service
- Task monitor
- microkernel SPARTAN
- Console server
- Framebuffer service
Making it all hold together
Making it all hold together

- The previous slide shows 28 independent server tasks running at one time
Making it all hold together

- The previous slide shows 28 independent server tasks running at one time
- Some servers may even run in multiple instances
Making it all hold together

- The previous slide shows 28 independent server tasks running at one time
- Some servers may even run in multiple instances
- All servers provide some services to other server tasks or applications; most servers require services from other servers
Making it all hold together

- The previous slide shows 28 independent server tasks running at one time
- Some servers may even run in multiple instances
- All servers provide some services to other server tasks or applications; most servers require services from other servers
- Together these server tasks provide the services of the operating system
Making it all hold together (II)

- So how do these tasks communicate?
Making it all hold together (II)

- So how do these tasks communicate?
- Both the monolithic OS and the single system task microkernel OS deal only with one address space
Making it all hold together (II)

- So how do these tasks communicate?
- Both the monolithic OS and the single system task microkernel OS deal only with one address space
- In a multiserver OS, the servers are in separate address spaces
Making it all hold together (II)

- So how do these tasks communicate?
- Both the monolithic OS and the single system task microkernel OS deal only with one address space
- In a multiserver OS, the servers are in separate address spaces
- Message passing provided by the kernel
  - IPC
HelenOS IPC
HelenOS IPC

- Message passing
HelenOS IPC

- Message passing
  - unusual metaphor of making phone calls and leaving a message in the answerbox
HelenOS IPC

- Message passing
  - unusual metaphor of making phone calls and leaving a message in the answerbox
  - Asynchronous
HelenOS IPC

- Message passing
  - unusual metaphor of making phone calls and leaving a message in the answerbox
  - Asynchronous
- Number of communicating tasks can be 1, 2 or N
HelenOS IPC

- Message passing
  - unusual metaphor of making phone calls and leaving a message in the answerbox
    - Asynchronous
- Number of communicating tasks can be 1, 2 or N
  - communicating with self
  - communicating with a peer
  - peer forwards the call to third party
HelenOS IPC (II)

- Message ~ Phone call
  - simple calls
  - combo calls
HelenOS IPC (II)

- Message ~ Phone call
  - simple calls
  - combo calls
- Simple calls
  - Six 32-bit / 64-bit words of payload
HelenOS IPC (II)

- Message ~ Phone call
  - simple calls
  - combo calls
- Simple calls
  - Six 32-bit / 64-bit words of payload
- Combo calls
  - memory sharing
  - large data block copying
  - tasks negotiate, kernel arbitrates
Life with IPC
Life with IPC

- Restricting interactions between logical components to IPC has some advantages
Life with IPC

- Restricting interactions between logical components to IPC has some advantages
  - the components understand a protocol
  - the protocol can be verified
  - the protocol can have many implementations
Life with IPC

- Restricting interactions between logical components to IPC has some advantages
  - the components understand a protocol
  - the protocol can be verified
  - the protocol can have many implementations
    - object oriented design
Life with IPC (II)

- It also brings some problems
Life with IPC (II)

- It also brings some problems
  - writing IPC by hand is tedious compared to mere function calls in monolithic designs
Life with IPC (II)

- It also brings some problems
  - writing IPC by hand is tedious compared to mere function calls in monolithic designs
    - could be generated from some high level architecture description
    - all HelenOS IPC written by hand so far
Life with IPC (II)

- It also brings some problems
  - writing IPC by hand is tedious compared to mere function calls in monolithic designs
    - could be generated from some high level architecture description
    - all HelenOS IPC written by hand so far
  - it is difficult to implement non-trivial protocols using asynchronous IPC
Life with IPC (II)

• It also brings some problems
  – writing IPC by hand is tedious compared to mere function calls in monolithic designs
    • could be generated from some high level architecture description
    • all HelenOS IPC written by hand so far
  – it is difficult to implement non-trivial protocols using asynchronous IPC
    • callbacks and event loops
    • HelenOS has a framework for it
Asynchronous framework
Asynchronous framework

- Makes the asynchronous communication a pleasant experience
  - no event loops
  - no callbacks
Asynchronous framework

- Makes the asynchronous communication a pleasant experience
  - no event loops
  - no callbacks
- Introduces fibrils (userspace threads) to already multithreaded tasks
  - client's connection handled by a fibril in server
  - fibril can send asynchronous messages and wait for them later
Asynchronous framework (II)
Asynchronous framework (II)
Using async framework
Using async framework

- Waiting for a request
  - callid = async_get_call(&call)
Using async framework

- Waiting for a request
  - callid = async_get_call(&call)
- Answer a request with \( n \) return values
  - ipc_answer_\( n \)(callid, retval, ...)
Using async framework

- Waiting for a request
  - callid = async_get_call(&call)
- Answer a request with $n$ return values
  - ipc_answer_n(callid, retval, ...)
- Send one message with $n$ arguments
  - msgid = async_send_n(phone, method, ..., &answer)
Using async framework

- Waiting for a request
  - callid = async_get_call(&call)
- Answer a request with $n$ return values
  - ipc_answer_n(callid, retval, ...)
- Send one message with $n$ arguments
  - msgid = async_send_n(phone, method, ..., &answer)
- Wait for an answer to a sent message
  - async_wait_for(msgid, &retval0)
Using async framework (II)

- Send of $n$ arguments and receive of $m$ return values combined
  - $\text{retval0} = \text{async\_req\_n\_m}(\text{phone}, \text{method}, ..., ..., ...)$
Using async framework (II)

- Send of $n$ arguments and receive of $m$ return values combined
  - $\text{retval0} = \text{async\_req\_n\_m}(\text{phone}, \text{method}, ..., ...)$

- Sharing memory
  - $\text{async\_share\_in\_out\_start}(\text{phone}, ...)$
  - $\text{async\_share\_in\_out\_receive}(\text{&callid}, ...)$
  - $\text{async\_share\_in\_out\_finalize}(\text{callid}, ...)$
Using async framework (III)

- Copying data
  - `async_data_read/write_start(phone, ...)`
  - `async_data_read/write_receive(&callid, ...)`
  - `async_data_read/write_finalize(callid, ...)`
Using async framework (III)

- Copying data
  - async_data_read/write_start(phone, ...)
  - async_data_read/write_receive(&callid, ...)
  - async_data_read/write_finalize(callid, ...)

- Fibrils often need to be synchronized
  - Fibril synchronization primitives
    - Mutexes
    - Readers-Write locks
    - Condition variables
req = async_send_2(vfs_phone, VFS_IN_MOUNT, dev_handle, flags, NULL);
rc = async_data_write_start(vfs_phone, (void *) mpa, mpa_size);
if (rc != EOK) {
    ...
}
rc = async_data_write_start(vfs_phone, (void *) opts, str_size(opts));
if (rc != EOK) {
    ...
}
rc = async_data_write_start(vfs_phone, (void *) fs_name, str_size(fs_name));
if (rc != EOK) {
    ...
}
/* Ask VFS whether it likes fs_name. */
rc = async_req_0_0(vfs_phone, IPC_M_PING);
if (rc != EOK) {
    ...
}
async_wait_for(req, &rc);
if (read)
    res = async_data_read_receive(&callid, NULL);
else
    res = async_data_write_receive(&callid, NULL);
if (read)
    fibril_rwlock_read_lock(&file->node->contents_rwlock);
else
    fibril_rwlock_write_lock(&file->node->contents_rwlock);
msg = async_send_3(fs_phone, read ? VFS_OUT_READ : VFS_OUT_WRITE, 
    file->node->dev_handle, file->node->index, file->pos, &answer);
ipc_forward_fast(callid, fs_phone, 0, 0, 0, IPC_FF_ROUTE_FROM_ME);
async_wait_for(msg, &rc);
if (read)
    fibril_rwlock_read_unlock(&file->node->contents_rwlock);
else
    fibril_rwlock_write_unlock(&file->node->contents_rwlock);
ipc_answer_1(rid, rc, bytes);
if (read)
    res = async_data_read_receive(&callid, NULL);
else
    res = async_data_write_receive(&callid, NULL);
if (read)
    fibril_rwlock_read_lock(&file->node->contents_rwlock);
else
    fibril_rwlock_write_lock(&file->node->contents_rwlock);
msg = async_send_3(fs_phone, read ? VFS_OUT_READ : VFS_OUT_WRITE,
      file->node->dev_handle, file->node->index, file->pos, &answer);
ipc_forward_fast(callid, fs_phone, 0, 0, 0, IPC_FF_ROUTE_FROM_ME);
async_wait_for(msg, &rc);
if (read)
    fibril_rwlock_read_unlock(&file->node->contents_rwlock);
else
    fibril_rwlock_write_unlock(&file->node->contents_rwlock);
ipc_answer_1(rid, rc, bytes);
Code example (II)

- VFS
- libc
- TMPFS
Code example (II)

1. VFS_IN_READ

libc

VFS

TMPFS
Code example (II)

1. VFS_IN_READ

2. IPC_M_DATA_READ

VFS

libc

TMPFS
Code example (II)

1. VFS_IN_READ
2. IPC_M_DATA_READ
3. VFS_OUT_READ
Code example (II)

1. VFS_IN_READ
2. IPC_M_DATA_READ
3. VFS_OUT_READ
4. ipc_forward()
Code example (II)

1. VFS_IN_READ
2. IPC_M_DATA_READ
3. VFS_OUT_READ
4. `ipc_forward()`
5. `ipc_answer(EOK)`
Code example (II)

```
1. VFS_IN_READ
2. IPC_M_DATA_READ
3. VFS_OUT_READ
4. ipc_forward()
5. ipc_answer(EOK)
6. ipc_answer(EOK)
```
Code example (II)

1. VFS_IN_READ
7. ipc_answer(EOK)

2. IPC_M_DATA_READ

3. VFS_OUT_READ
6. ipc_answer(EOK)

4. ipc_forward()

5. ipc_answer(EOK)

VFS

TMPFS

libc
Demo
Questions?
www.helenos.org

jakub@jermar.eu