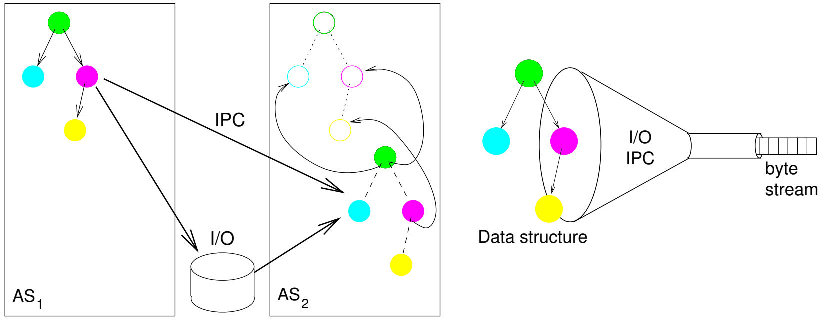
MUNGI

A SINGLE-ADDRESS-SPACE OPERATING SYSTEM

MOTIVATION: Use wide address spaces to

- ➜ increase efficiency,
- ➜ simplify programming.

Sharing in traditional systems:

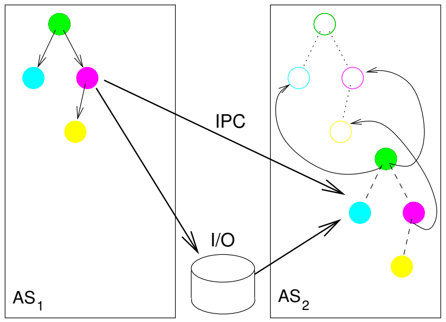


MUNGI: SINGLE-ADDRESS-SPACE OPERATING SYSTEM

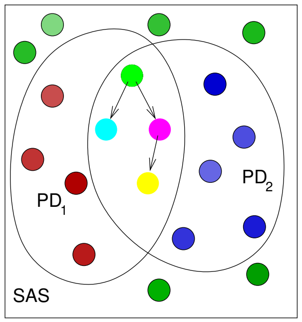
MOTIVATION: Use wide address spaces to

- ➜ increase efficiency,
- ➜ simplify programming.

Sharing in a traditional system:



in a single address space:



NAMING IN TRADITIONAL SYSTEMS:

memory data:

```
item_t a, *x;

...
a = *x;
```

address is `x`

file data:

```
item_t a;
int    x;
FILE   *f;
...
f = fopen("f","r");
fseek (f, x, SEEK_SET);
fread (*a, sizeof(item_t), 1, f);
```

address is `("f",*x)`

Result: complex programs

NAMING IN SINGLE-ADDRESS-SPACE SYSTEMS:

memory data:

```
item_t a, *x;

...
a = *x;
```

address is `*x`

Result: simplified programs

SASOS advantages: application view

- Simple naming mechanism – 64 bit address – supported by "conventional" hardware.
- User data structures can contain embedded references to other data.
- Eliminates excessive copying of data and software pointer translation.

SASOS advantages: system view

- Simplifies data migration
- Simplifies process migration
- orthogonality of translation and protection
- no need for file system — all disk I/O is paging
- RAM is cache for VM — unified buffer & disk cache management
- easy to implement zero-copy operations
- in-place execution — no need for position-independent code

⇒ Simplified system implementation and increased performance

SASOS advantages: hardware view

- Virtual caches are no problem
  virtual address maps uniquely to physical address
- Split TLBs are neat
  cannot otherwise distinguish between R and X access
- hardware separating translation from protection could increase performance due to increased TLB coverage
  (AS/400 segment lookaside buffer, IA-64 protection keys)

Single-Address-Space Operating Systems

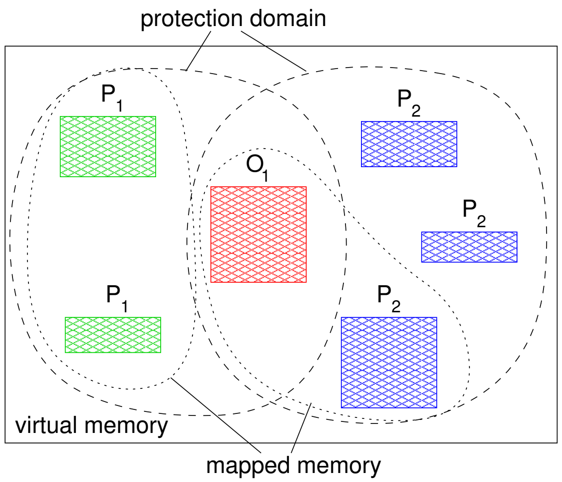
IBM System/38 [Ber80] & AS/400 [Sol96] 1978 (at the lowest level)
- ➜ heavyweight object model,
- ➜ requires special hardware.

Angel [MSS+93, MSWK94, Wil93, WM96] 1992–5 (City U, London):
- ➜ 32-bit prototype,
- ➜ performance?

Opal [CLBHL92, CLFL94] 1992–4 (University of Washington):
- ➜ prototype on top of Mach,
- ➜ poor performance.

Mungi [HEV+98] (UNSW): Goals
- ➜ push single-address-space model to its limits,
- ➜ demonstrate excellent performance,
- ➜ explore potential benefits to traditional systems.

PROTECTION IN A SASOS



PROTECTION:

- Everything is *visible*.
- *Protection domain* defines what is *accessible*.
- Access requires mapping virtual to physical addresses.
- Mapping established by system.

⇒ System controls access by establishing *partial view* of the single address space.

- Can implement usual protection models (ACLs, capabilities).

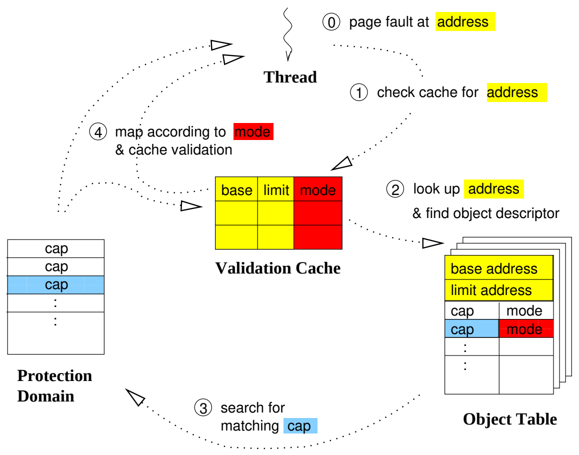
MUNGI'S PROTECTION MODEL:

- *Memory object* (contiguous set of pages) is *unit of protection*.
- Each object has an associated set of *password capabilities*
  Capabilities give some combination of `rwxd` rights.
- *Protection domain objects* (PDOs) used to define access rights.
- *Threads* execute in an *active protection domain* (APD) instantiated from a PDO.
- A new thread can:
  - ➜ run in parent's APD,
  - ➜ run in new APD specified by parent.

PROTECTION DOMAINS:

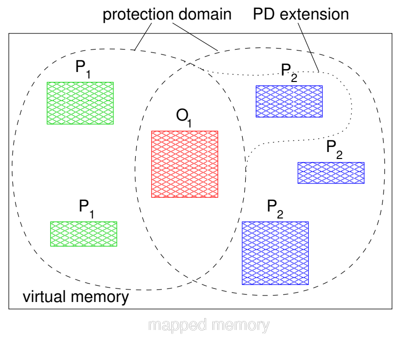
- A **PDO** is a *special* object:
  - ➜ system-defined structure,
  - ➜ PD attribute allows use in thread creation,
  - ➜ consists of an array of *clist* capabilities,
  - ➜ read/write access restricted to some "privileged" code.
- A **clist** is a "normal" object:
  - ➜ contents under user control (permissions allowing...),
  - ➜ **but** with system-defined structure,
  - ➜ users deposit their capabilities in clists.
- A new APD is created by copying a PDO to the TCB.
- On validation, the clists are searched by the kernel
  - ➜ Capability presentation is *implicit*.

Access validation



Protected Procedure Calls

- Objects can have *protection-domain extension* (PDX) type:
  - ➜ have registered set of *entry points*,
  - ➜ have *PDX capabilities*,
  - ➜ have an associated *clist*.
- Holder of PDX capability can invoke entrypoint of PDX object.
  - ➜ This changes the invoker's APD *for the duration of the call*.

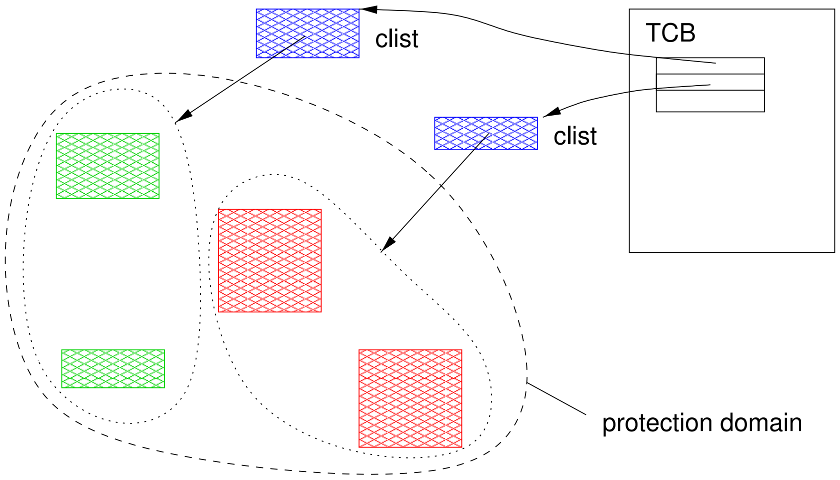


Protection domain manipulation:

- All capability presentation is *implicit* (via clists).
- A thread can manipulate its protection domain:
  - ➜ by modifying its clists,
  - ➜ **provided that the APD contains the clists**.
- A thread can be set up so that it's APD:
  - – does not contain the clists defining it,
  - – does not contain write access to any "public" objects.

  Such a thread is *confined*.

Confinement



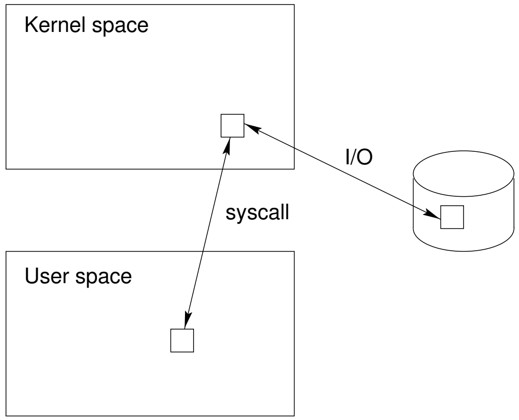
MUNGI PHILOSOPHY

- Virtual memory is everything:
  - ➜ No message-passing IPC supported.
  - ➜ No I/O abstraction supported.
- Everything is virtual memory:
  - ➜ OS data structures are inside the global address space.
  - ➜ This should help distribution.

HOW ABOUT DEVICES?

- Mungi has **no** I/O model.
- Devices can be
  - ➜ mapped into virtual memory, or
  - ➜ accessed via a procedural interface (using a PDX procedure).
- Actual device model is under construction.

HOW ABOUT FILES?

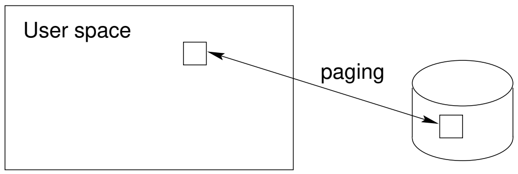


TRADITIONAL FILE SYSTEMS...

... require:

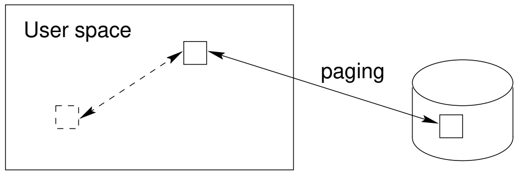
- explicit system invocation
- multiple copying

I/O in Mungi



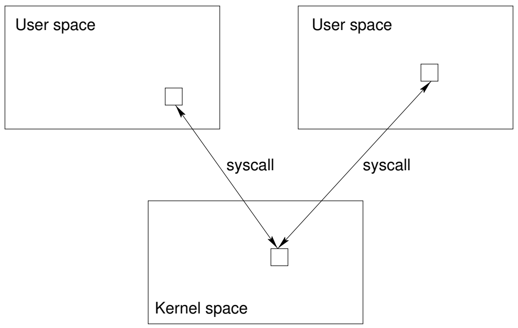
... eliminates overhead

I/O in Mungi...



feel free to copy...

How about IPC?

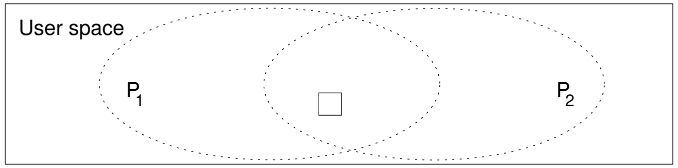


IPC in traditional systems...

... requires:

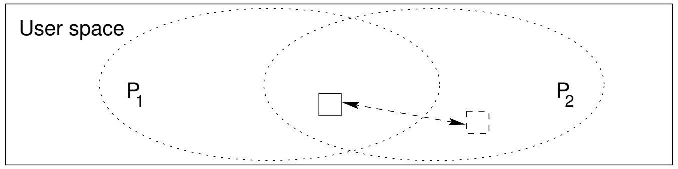
- explicit system invocation
- multiple copying

IPC in Mungi



eliminates overhead

IPC in Mungi



feel free to copy...

How About POSIX Compatibility

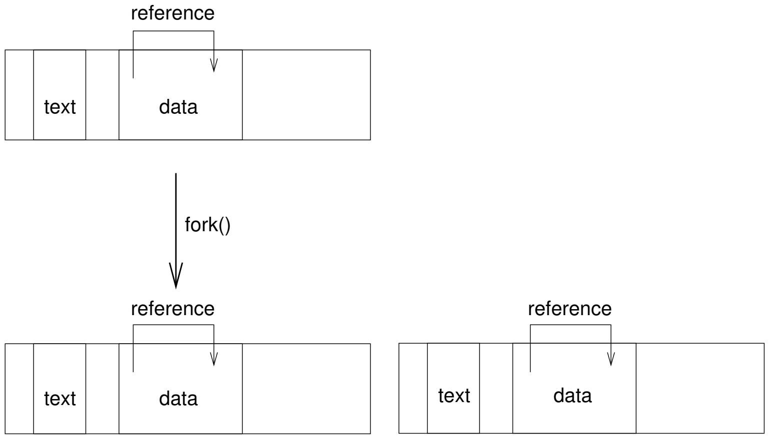
- Revolutionary new systems have little chance of acceptance if they cannot emulate the traditional world

⇒ need POSIX emulation

I/O easy: memory file system:

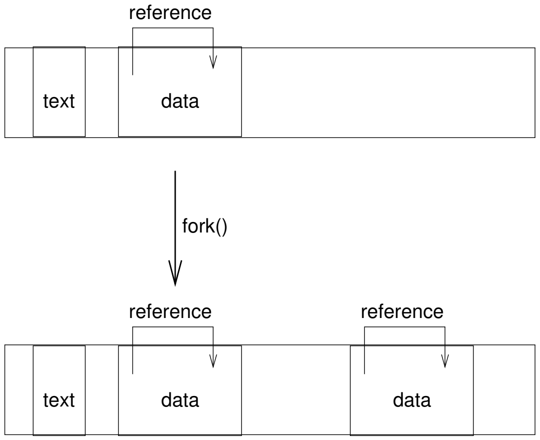
- open gets object address from name server sets up descriptor
- read/write directly write to object without explicitly buffering
- read-ahead can be implemented by having a thread touch next page

POSIX fork()



Duplicates address space

SASOS FORK()



Relocates within address space [WMSS93]

MEASURED COST OF FORK() SUPPORT

	OO1	*heapsort*	*nsieve*
normal	33.9	37.8	196.8
forkable	34.6	38.0	196.8
overhead	2.0 %	0.5 %	0.0 %

On MIPS R4600

Notes:

➜ Overhead only affects programs insisting on full POSIX compliance.

➜ fork() overhead fully absorbed by that of dynamic linking [DH99].

MUNGI IMPLEMENTATION

PROTOTYPE RUNNING ON MIPS R4600 AND ALPHA 21X64 CPUS:

- Implemented on top of the L4 microkernel.
- Mungi consists of small number of L4 tasks.
- One (or more) L4 task per APD.
- New threads can be started in current APD
  ➜ local L4 thread.
- New threads can be started in specified PD
  ➜ new L4 task.
- Most of Mungi itself runs in single address space.
- Still rudimentary.

[VRH93, VERH96, HEV+98, Voc98, HVER97]

MUNGI PERFORMANCE

Operation	*Mungi*	*Linux*	*Irix*	*Opal*
Microbenchmarks				
Null system call	4.6	6.3	7.7	>88
Cross-domain call	10–20	161	450	133
Thread create+delete	131/96	2,450	4,882	N/A
Task create+delete	910	34,163	8,150	2,950
Object access	90	45	252	239?
Object create+access+delete	300	447	2470	2,100
OO1 (in memory)				
total time	189	2,114	3,972	N/A

References

[Ber80] Viktors Berstis. Security and protection in the IBM System/38. In *Proc. 7th Symp. Comp. Arch.*, pages 245–250. ACM/IEEE, May 1980.

[CLBHL92] Jeff S. Chase, Hank M. Levy, Miche Baker-Harvey, and Edward D. Lazowska. Opal: A single address space system for 64-bit architectures. In *Proc. 3rd W. Workstation Operating Syst.*, pages 80–85, Key Biscayne, FL, USA, 1992. IEEE.

[CLFL94] Jeffrey S. Chase, Henry M. Levy, Michael J. Feeley, and Edward D. Lazowska. Sharing and protection in a single-address-space operating system. *ACM Trans. Comp. Syst.*, 12:271–307, November 1994.

[DH99] Luke Deller and Gernot Heiser. Linking programs in a single address space. In *Proc. 1999 USENIX Technical Conf.*, pages 283–294, Monterey, Ca, USA, June 1999.

[HEV+98] Gernot Heiser, Kevin Elphinstone, Jerry Vochteloo, Stephen Russell, and Jochen Liedtke. The Mungi single-address-space operating system. *Softw.: Pract. and Experience*, 28(9):901–928, July 1998.

[HVER97] Gernot Heiser, Jerry Vochteloo, Kevin Elphinstone, and Stephen Russell. The Mungi kernel API/Release 1.0. Technical Report UNSW-CSE-TR-9701, School Comp. Science and Engin., University NSW, Sydney 2052, Australia, March 1997. Latest version available from http://www.cse.unsw.edu.au/~disy/.

[MSS+93] Kevin Murray, Ashley Saulsbury, Tom Stiemerling, Tim Wilkinson, Paul Kelly, and Peter Osmon. Design and implementation of an object-orientated 64-bit single address space microkernel. In *Proc. 2nd USENIX W. Microkernels and other Kernel Arch.*, pages 31–43, September 1993.

[MSWK94] Kevin Murray, Tom Stiemerling, Tim Wilkinson, and Paul Kelly. Angel: Resource unification in a 64-bit micro-kernel. In *hicss94*, 1994.

[Sol96] Frank G. Soltis. *Inside the AS/400*. Duke Press, Loveland, CO, USA, 1996.

[VERH96] Jerry Vochteloo, Kevin Elphinstone, Stephen Russell, and Gernot Heiser. Protection domain extensions in Mungi. In *Proc. 5th IEEE IWOOOS*, pages 161–165, Seattle, WA, USA, October 1996. IEEE.

[Voc98] Jerry Vochteloo. *Design, Implementation and Performance of Protection in the Mungi Single-Address-Space Operating System*. Phd thesis, School Comp. Science and Engin., University NSW, Sydney 2052, Australia, July 1998. Available from http://www.cse.unsw.edu.au/~disy/papers/.

[VRH93] Jerry Vochteloo, Stephen Russell, and Gernot Heiser.

Capability-based protection in the Mungi operating system. In *Proc. 3rd IEEE IWOOOS*, pages 108–15, Asheville, NC, USA, December 1993. IEEE.

[Wil93] Tim Wilkinson. *Implementing Fault Tolerance in a 64-Bit Distributed Operating System*. PhD thesis, Systems Architecture Research Centre, City University, London, UK, July 1993.

[WM96] Tim Wilkinson and Kevin Murray. Evaluation of a distributed single address space operating system. In *Proc. 16th IEEE ICDCS*, pages 494–501, Hong Kong, May 1996. IEEE.

[WMR+95] Tim Wilkinson, Kevin Murray, Stephen Russell, Gernot Heiser, and Jochen Liedtke. Single address space operating systems. Technical Report UNSW-CSE-TR-9504, University NSW, University NSW, Sydney 2052, Australia, November 1995.

[WMSS93] Tim Wilkinson, Kevin Murray, Ashley Saulsbury, and Tom

Stiemerling. Compiling for a 64-bit single address space architecture. Technical report TCU/SARC/1993/1, Systems Architecture Research Centre, City University, London, UK, March 1993.

