The Mungi Kernel API

Version 1.2

August 28, 2002

disy@cse.unsw.edu.au
http://www.cse.unsw.edu.au/~disy/
Operating Systems and Distributed Systems Group
School of Computer Science and Engineering
The University of New South Wales
UNSW Sydney 2052, Australia

Abstract

This document describes version 1.2 of the application programming interface to the kernel of the *Mungi* single-address-space operating system. This interface will, in general, only be used by low-level software, most applications are expected to use a higher-level interface implemented as system libraries. Such libraries will be described in separate documents.

Contents

Chapter 1

Introduction

Mungi [RSE+92, HERH93, VRH93, HERV94, ERHL96, VERH96, HEV+98, DH99] is a single-address-space operating system (SASOS) [CLFL94, WMR+95] developed by the Operating Systems and Distributed Systems Group at the University of New South Wales. It is conceptually similar to Angel [WSO+92, WM96] and Opal [CLBHL92, CLFL94], although quite different in many important aspects. Most notably, Mungi presents the single-address-space model in a very pure form, as it provides no inter-process-communication mechanisms other than shared memory. Many of the basic ideas in Mungi go back to the IBM System/38 [Ber80, HSH81] and its successor, the AS/400 [Sol96].

This document presents the application programming interface (API) of the Mungi kernel. While Mungi does not pretend to be a microkernel (in fact, the prototype is implemented on top of the L4 microkernel [Lie95]), it nevertheless presents a minimal and low-level interface to the programmer. Policies are, as far as possible, left to be implemented by higher software layers. Similar to microkernels, Mungi allows the implementation of device drivers and page fault handlers at user level.

As a consequence, application programs will not normally interface directly to the Mungi kernel, but are expected to call a higher level library interface. A UNIX-like interface has partially been developed, however this will be described in a separate document.

The following sections list and explain the Mungi system calls. These are presented in (hopefully) intuitive pseudocode. The actual C language bindings are presented in the Appendix.

1.1 Conventions

This document presents Mungi system calls and data structures in an abstract format, showing the main characteristics but not detailed type information etc. Consult the C language bindings in Appendix A for actual types and syscall signatures.

The system call tables show possible failure modes of various calls. An error condition is indicated by the syscall returning a value indicating failure, the syscall *GetLastError()* can then be used to obtain the actual error number.

Chapter 2

Objects and Capabilities

Objects are Mungi's storage abstraction, they are the unit of virtual memory allocation and protection. Objects are page aligned and consist of an integral number of pages. Newly allocated objects are zero filled. From the system's point of view, an object is simply a contiguous, aligned region of virtual address space; the system imposes no structure on objects (but higher software levels are free to do so). Access to objects is controlled by *password capabilities* [APW86].

2.1 Password capabilities

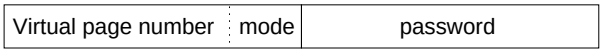

Figure 2.1: Format of a capability

Figure 2.1 shows the format of a capability. The virtual page number refers to the base address of an object. The mode indicates the rights conferred by the capability. This field is only a hint for the user, it is ignored by the system when a capability is presented. The system maintains a directory, the *object table* (OT), of object attributes, including the set of valid passwords and their corresponding modes.

There are five different rights capabilities may grant over an object: read (R), write (W), execute (X), destroy (D), and *protection domain extension* (PDX); the latter will be explained in Section 3.3. Each valid capability grants the holder a combination of these rights to an object.[1] A capability granting RWXD rights is, by definition, an owner capability. A capability containing a zero password is considered a valid capability granting no access rights.

Capabilities can confer negative rights (*negative capabilities*). These can be used to ensure that a client does **not** have certain access rights. For example if, while validating an attempted write access, the system encounters a *not write* capability, it will raise a protection fault. A negative capability is indicated by a *NOT* bit in the mode field. If this bit is set, the capability negates the rights indicated by the other rights bits.

Capabilities are user objects and can be stored and passed around freely. They are protected from forgery by sparsity, hence careful choice of passwords is important. The system provides a library routine to create random passwords.

As capabilities are user objects, it is not possible to determine who has access to a particular object. It is also normally impossible to prevent a particular user, who has been given a capability for an object,

[1] Note that, as we are relying on the hardware to enforce protection, on many architectures we cannot guarantee that a user cannot read an object to which they only hold an X capability.

from handing this capability to other users. However, the *ObjPasswd* call allows revocation of a capability. Furthermore there exist a mechanism for confining clients of one's objects, see Section 3.4.

2.2 Special objects

Some objects are of a *special* nature in the sense that certain system calls require a reference to such objects, and the system must be able to guarantee that such objects can only be created by authorised agents.

Presently there are two types of special objects: *bank accounts* and *protection domains*. The former constitute the basic mechanism for resource accounting in Mungi, while the latter support creation of threads with access rights different from the caller's.

The use of bank accounts for resource management is discussed in [HLR98]. The support provided by Mungi is to make bank accounts special and to ensure that every object created has an associated bank account. The use of protection domains is further explained in Chapter 3.

What makes an object special is an association with a *controlling object*. The system protects speciality by only allowing a thread to make an object special *if the thread has read access to the controlling object*. The type of the controlling object must agree with the speciality type requested for the controlled object. Special objects whose controlling object no longer exists are cleaned up by the system lazily.

An object which is its own controlling object is a *master object* for a type of special objects. Creation of master objects requires special privilege (read access to the *object table*).

Special objects are not executable code, and the execute permission bit takes on a special meaning for these objects: It allows the holder to use the object (charge an object's storage to a bank account, start a thread in a protection domain) without being able to read the object's data.

2.3 Object descriptors

Figure 2.2 shows the format of an object descriptor which is used to describe object attributes. That descriptor format is used when inquiring or setting object attributes in the OT. The meaning of the fields is as

type *ObjInfo* $\equiv$ {
  comment: public part
  length, *extent*,
  create_time, *modify_time*, *access_time*, *accounting_time*,
  user_info, *accounting_info*,
  comment: restricted part
  {*password*, *mode*}[*n_caps*],
  {*clist_cap*, *password*, *entry_pt*[*n_entry*]}[*n_pdx*],
  bank_acct_cap, *pager_cap*, *flags*,
  special, *controlling_cap*
}

Figure 2.2: Object descriptor (*info*) data structure.

follows:

length: length of the object in bytes. The system will round this to a multiple of the basic hardware page size (or bigger) and will also take into account the *extent* value.

extent: unit of backing store allocation. The system rounds this to an integer multiple of the page size (which may be the next bigger integer multiple, or the next bigger power-of-two multiple). The system may ignore changes to the original setting.

create_time, *modify_time*, *access_time*: timestamp of object creation, last modification, last access. The modification and access time stamps are automatically updated by the system (with limited accuracy).

accounting_time: timestamp used by the accounting system.

accounting_info: reserved for usage by accounting system.

user_info a user-defined field which may be used, e.g., for implementing a type system on top of the Mungi kernel. The system does not use this field, but ensures that only an owner can change it.

password, mode: list of non-PDX passwords plus access rights conferred by each password.

clist_cap: Clist capability for extending the APD when the object is invoked via the *PdxCall* system call (c.f. Section 3.3).

password, entry_pt[]: list of passwords allowing PDX call, plus the set of entrypoints allowed for each password.

bank_acct_cap: R capability to the *bank account* to which storage costs for the object is charged. Bank accounts are the basis of Mungi's management of memory resources. The kernel only provides the mechanism by ensuring that a bank account is associated with every object, and by protecting bank accounts from forgery. The operation of the accounting system is described in [HLR98].

pager_cap: PDX capability to the page fault handler for the object. Null if handled by the default pager.

flags: array of other object attributes. Present values:

persistent: indicates that the object may outlive its creator. An object created with this flag off will be cleaned up by the system when the creator thread exits. The flag can be turned on by an owner, however, this may have no effect if called from a thread running in an APD different from the creator's. The flag can be turned off by an owner, in which case the object is marked for cleanup when the caller's thread exits. Object cleanup may be delayed if the exiting thread is the descendent of another thread running in the same APD.

special: array of flags identifying special object types. An attempt to change any of these will only be honoured if the controlling object is not-null (or being made not-null at the same time), the caller has read access to the controlling object and the controlling object has the corresponding bit set. Present values:

is_acct: indicates that the object can be used as a *bank account*.

is_financial: indicates that the bank account object can presently be used for object creation.

is_pd: indicates that the object can be used as a *protection domain*.

controlling_cap: specifies that the object is *special* and has the specified *controlling object*. An attempt to change the controlling object attribute will only be honoured if the caller has read access to that controlling object and the latter is *special*. An object can be made its own controlling object if the caller has read access to the object table.

2.4 System calls

System calls dealing with objects are summarised in Table 2.1 and explained below.

System calls:	
ObjCreate	(*size, password, info*) → (obj_adr)
	fails: *invalid_info, out_of_memory, invalid_bank_account*
ObjDelete	(*adr*)
	fails: *protection_violation*
ObjResize	(*adr, new_size*)
	fails: *protection_violation, invalid_info, cannot_grow, out_of_memory*
ObjNewPager	(*adr, pager*)
	fails: *protection_violation, invalid_PDX*
ObjPasswd	(*cap, mode*)
	fails: *protection_violation, table_overflow*
ObjCrePdx	(*cap, clist, n_entries, entry[]*)
	fails: *protection_violation, table_overflow, invalid_PDX, invalid_Clist, invalid_NULL_value*
ObjInfo	(*adr, update_flags*, OT_entry) → (*OT_entry*)
	fails: *protection_violation, invalid_info*

Table 2.1: System calls dealing with objects and capabilities.

ObjCreate: allocates a new object of a specified *size*. The call returns the base address of the object. Together with the password supplied by the caller, this will make an *owner capability* for the new object. The caller supplies an optional *ObjInfo* data structure (to which they must have read access) which is used to initialise the object descriptor in the OT. The fields used are *bank_acct_cap*, *extent*, *user_info*, *flags*, *special* and *controlling_cap*, other entries are ignored. Setting *special* and *controlling_cap* requires special privilege as explained above. If the *bank_acct_cap* is not given (or no *info* parameter supplied) the caller's default bank account is used.

The system may use some of the data in *info* as a hint to optimise its allocation strategy.

Initially, all page faults on the object are handled by the *default pager*, which initialises pages to zero. Note that a newly created object is not accessible until entered into a Clist.

ObjDelete: deallocates the object containing *adr*. The caller's protection domain must contain a destroy capability for the object, or must have execute access on the object's bank account. Virtual memory previously allocated to that object may be reused for objects created in the future, hence it is important that passwords cannot be guessed. Reuse of virtual memory will only happen once it is certain that no validations are still cached.

ObjResize: expands or shrinks the object containing *adr* to *new_size*. An object may not be able to grow if the following address space has already been allocated to a different object; a *cannot grow* exception will be raised in this case. The caller's protection domain must contain an owner capability for the object.

ObjNewPager: registers a new page fault handler for the object containing *adr*. A null *pager* address reverts to the system's default pager. The caller's protection domain must contain an owner capability for the

object. Executing this system call implies calling *PageUnmap(base,length,zero)* on the object. See Chapter 5 for details.

ObjPasswd: registers a new capability for an object, or removes an existing one for the object referred to by *cap*. A capability is deleted by specifying its password and a null *mode*. The *mode* parameter specifies the strength of that capability. If *mode* is RWXD, the new capability is an owner capability, conferring the same rights as the original owner capability. The PDX bit, if set in *mode*, is ignored. The caller's protection domain must contain an owner capability for the object.

ObjCrePdx: registers a new PDX capability for the object referred to by *cap*, together with the protection domain extension (one Clist) and a list of valid entry points for which that password can be used. The caller's protection domain must contain an owner capability for the object, and at least execute rights to the Clist.

If a PDX password for this object has been registered before, *clist_cap* will overwrite any protection domain extension previously registered, except if null, in which case the protection domain extension remains unchanged. If the same PDX password has been registered for the object before, the list of entry points for that password will be reset to the one specified. However, if *n_entries* is negative, the list of entry points will not be changed. If *n_entries* is zero, the password will be revoked as a PDX password.

The call fails if no protection domain extension is registered at all, i.e. if an attempt is made to register the first PDX password with *clist_cap* being null or *n_entries* being less than or equal to zero.

PDX procedures are described further in Section 3.3.

ObjInfo: updates the object descriptor in the OT for the object containing *adr*, requires at least read or execute rights on the object. The system ignores all passwords as well as the *pager* field. *update_flags* specify which attributes are to be updated. An attempt to set *extent* may be ignored. Returns the previous value of the object descriptor. Passwords are returned if called by an owner, otherwise passwords will be returned as zeros.

The *controlling_cap* is only set if *all* of the following conditions are met:

- the controlling capability was null before the call, or the caller has read access to the old controlling object,
- the new controlling capability is null or the caller has read access to the new controlling object,
- the new controlling object is of the same *special* type as the target object is to become.[2]

If the caller has owner rights, *bank_acct_cap*, if given, is validated and used as the new bank account, otherwise this field is also ignored.

Time stamps may be updated by setting an appropriate bit in *update_flags*. The table below gives the minimum amount of privilege required for the various operations possible. Here *touch* means setting a time stamp to the present time and date, while *set* means setting to an arbitrary time and date. The *R-OT* privilege means that a R capability for the OT is required (note that this implies RWXD capability on any object). As usual, *RWXD* stand for an owner capability, *W* is a capability which allows write access, while *R/X* means a capability allowing read *or* execute access. The creation timestamp cannot be changed.

[2] Note that it is not sufficient for *controlling_cap* to be valid for read access, the read access must be allowed by the caller's active protection domain see Section 3.2.

Time stamp	*min. privilege for*	
	touch	*set*
modification	W	RWXD
access	R/X	RWXD
accounting	R-OT	R-OT

Chapter 3

Protection Domains

In general, a thread's protection domain is its set of access rights to objects. In a capability system, this is equivalent to a set of capabilities.

Mungi's protection system has been designed to be unintrusive, hiding its operation from applications as far as possible. For this reason, Mungi does not expect capabilities to be presented explicitly when an object is accessed. Instead, users store their capabilities in a datastructure which is searched by the kernel when an access validation needs to be performed. This datastructure is called a thread's *active protection domain* (APD).

A new thread is created by the parent performing a *ThreadCreate* system call. That call takes a parameter *pd* which indicates the protection domain in which the new thread is to execute. If the *pd* parameter is null, the new thread *shares*, for its lifetime, the parent's protection domain. Otherwise the thread is created in its own APD. The APD is in this case instantiated from a template *protection domain object* referenced by the *pd* parameter. A flag determines whether the new thread may join the APD of another thread whose APD has been instantiated from the same PD object. The system ensures that, if it allocates a stack object for a thread running in a newly created APD, capabilities to the stack object are inserted into the APD (see Section 4.1 for details).

3.1 Protection domain objects

A *protection domain object* (PD object) is a special object with the *is_pd* flag set. It is used to describe a protection domain in which threads can be created. The structure of a PD object is:

$$\begin{array}{l}\underline{\textbf{type}}\ apddesc\ \equiv\ \{\\ \quad clist[n\_pd], n\_locked;\\ \}\end{array}$$

Here, *clist* is a capability for a *capability list* (Clist) and *n_locked* specifies the number of locked slots in the APD. The semantics of these fields is described in Sections 3.2 and 3.4 below. Note that the first entry in the *clist* array will be overwritten on use and should therefore be left empty.

When a thread is created with a valid non-null *pd* parameter, a new APD is created for the thread. The new APD is a copy of the PD object referenced by the *pd* parameter. For more details see Chapter 4.

3.2 Active protection domains

A thread's APD is represented by the same information as what is found in a PD object. The *apddesc* data structure is kept in kernel space as part of the internal thread description. The actual Clists are user objects (and, as such, themselves parts of protection domains), as shown in Figure 3.1.

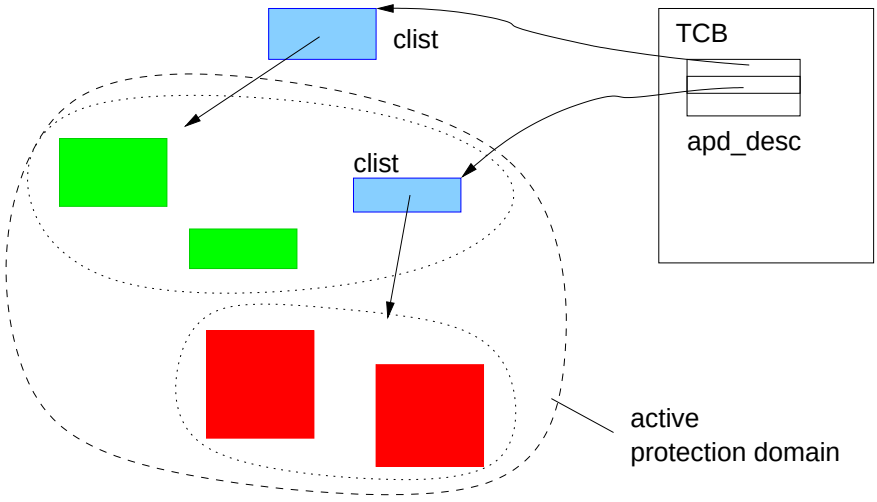

Figure 3.1: Active protection domain.

Clists are user-maintained datastructures in a standard format. Users can, protection permitting, add or remove capabilities in their Clists at any time without system intervention. Addition or removal of Clist capabilities to a running thread's APD is possible via the *ApdInsert* and *ApdDelete* system calls. Such an operation will affect all threads sharing an APD.

Validation of an access is normally performed by the system in response to a protection fault, i.e., an object was accessed for which the kernel does not have information on the validity of the access, or that information is inconsistent with the mode of the access. In order to perform the validation, the system searches the OT with the faulting address. If no matching object is found, a protection fault is signalled to the user thread. Otherwise, the APD is searched for a capability matching one of those in the OT with appropriate mode. If found, a mapping for the object is established and the validation information is cached by the kernel to avoid having to validate each page of a large object individually. The validation process is depicted in Figure 3.2. The basic algorithm is shown in Figure 3.3.

As the validation algorithm shows, the search is terminated when the first (positive or negative) capability of sufficient strength is found. Users can use this to arrange their capabilities such as to avoid double validation faults.

Presently there are two Clist formats, an unsorted one and one where capabilities are sorted by object base address. The kernel uses binary search on sorted Clists.

Any invalid (as opposed to negative) capabilities encountered while searching the APD are ignored. This avoids race conditions with newly created capabilities, and synchronisation problems problems if validation occurs concurrently with Clist updates. Searching a (due to updates) inconsistent "sorted" Clist may fail to locate an existing capability. Users are therefore responsible for setting the format indicator to "unsorted" before adding or removing capabilities in a Clist.[1] The search order *within* a Clist is undefined.

[1]Note that this scheme cannot completely prevent such failures. However, we expect that sorted Clists are modified very infrequently. The extremely rare chance of a transient failure to locate a capability does not seem to justify the expense of proper

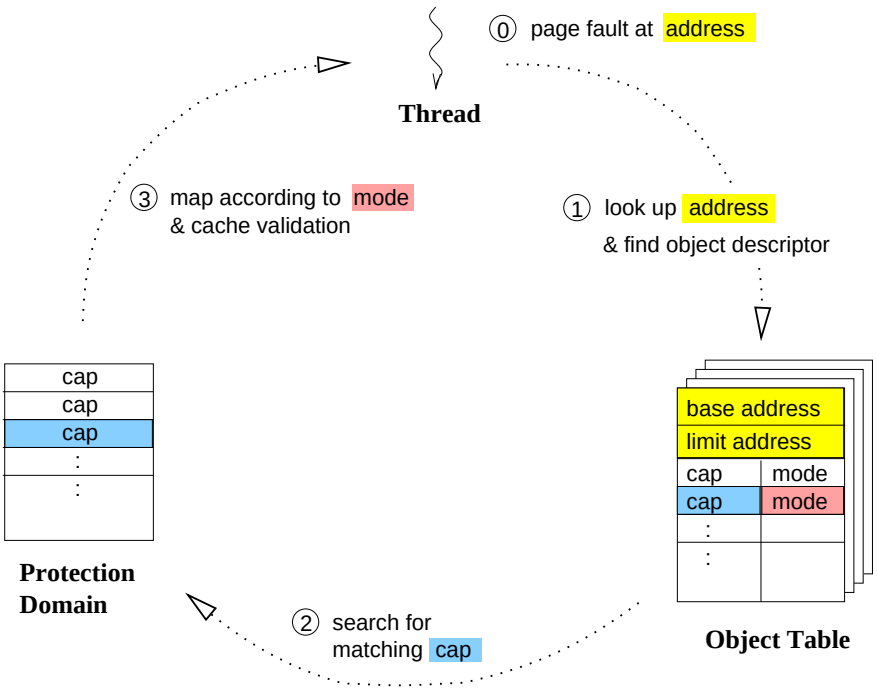

Figure 3.2: Access validation.

```
funct validate(fault_adr, mode) ≡
  if ¬OT_lookup(fault_adr) → (base, limit, caps) then
    raise protection violation;
  fi;
  for i = 0 to apd.n_pd-1 do
    foreach c ∈ apd.clist[i] do
      if c ∈ caps ∧ c.negative ∧ mode ⊇ c.mode then
        raise protection violation;
      elsif c ∈ caps ∧ mode ⊆ c.mode then
        return (base, limit, c.mode);
      fi;
    od;
  od;
  raise protection violation.
```

Figure 3.3: Access validation algorithm.

Caching of validations could delay the effect of revocations of passwords indefinitely. To prevent this, the system guarantees that no validations are cached for more than a certain amount of time. Clist capabilities in the APD are also periodically revalidated; if such a capability is found to have become invalid it is silently removed from the APD.

---

concurrency control. Paranoid users can, of course, replace the whole Clist, which can be done without ever leaving an inconsistent Clist in the APD.

3.3 Protected procedure calls

Mungi provides a protected procedure call mechanism similar to the *profile adoption* mechanism of the IBM System/38 [Ber80]. Mungi's mechanism, called *protection domain extension* (PDX) allows the caller of a PDX procedure to change its protection domain, for the duration of the call, in a controlled fashion.

More specific, a PDX procedure has, in the object table, registered a set of valid entry points and a capability for a Clist (c.f. Section 2.3). When a *PdxCall* system call is executed, the system first verifies that the caller possesses a valid PDX capability and tries to access a valid entry point, then extends the caller's APD by the PDX's Clist, and finally transfers control to the PDX code. When the PDX procedure returns, the PDX Clist (and all cached validation information relating to that Clist) is removed from the caller's APD. Note that for the duration of the PDX call, the calling thread's change of protection domain does not affect other threads executing in the caller's APD — such threads have no access to the called object (unless they also perform an appropriate PDX call).

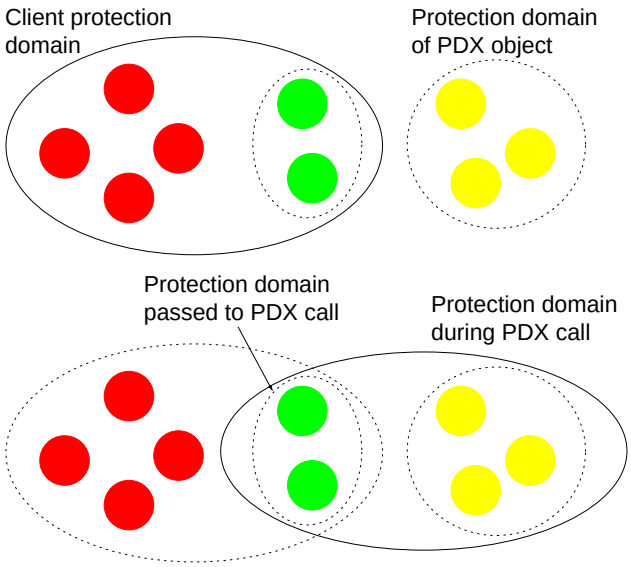

Figure 3.4: Active protection domains during a PDX call.

Instead of having the PDX procedure execute in a superset of the caller's protection domain, the caller has the option of explicitly supplying a protection domain object reference when calling the PDX procedure. In this case, the call executes in a protection domain which is the union of the supplied one with that registered for the PDX object. This is shown in Figure 3.4.

The ability of passing a protection domain gives the caller maximum control over which objects the PDX procedure can access. In particular, an empty protection domain may be passed to the PDX procedure, the latter then has no access to any of the caller's data (other than by-value parameters). Note that it is also possible to pass a capability (say for a result buffer) on the call explicitly; the PDX procedure must then insert that capability into one of its Clists.

3.4 Discretionary confinement

Since the APD data structure contains actual capabilities for Clists, there is no need for Clist capabilities to be contained in any of a thread's Clists. As Clist capabilities are immediately validated when added to

the APD, the system can rely on all Clists referenced by the APD to be accessible. A thread holding no read capability to any of its Clists can use the Clists to access objects, but cannot look at the capabilities contained in the Clists themselves.

Applications can make use of this to *confine* untrusted code. Mungi provides the facility of *locking* part or all of a thread's APD. Locked Clist capabilities cannot be changed or removed, and no new Clist capabilities can be added before them. This also applies to the (implicit) addition of Clist capabilities during a PDX call — PDX Clist capabilities will only be inserted into the caller's APD *after* any locked slots. A thread whose APD is defined by Clists which are all outside the APD, and whose APD is locked, has no way to modify its APD.

A thread with a locked APD, and with no write capabilities to objects readable by others, is unable to leak any of the data it has access to. A partially locked APD does not confine a thread, but can ensure that certain objects remain unaccessible, provided that one of the locked Clists contains sufficient *negative* capabilities to the objects which are not to be accessed [Voc98]. Note that this form of confinement can only work because all capability presentation is implicit.

The system libraries contain a procedure which, using one-way functions, generates reduced-strength capabilities from a given capability. While usage of this method is not enforced by the system, using it makes it easy to construct Clists containing only read-only (or execute-only) capabilities to globally used objects, without requiring the owners to distribute a whole set of capabilities for each object.

3.5 System calls

APD operations are summarised in Table 3.1. These calls work as follows:

System calls:		
ApdInsert	(*pos, clist_adr*)	
	fails: *protection_violation, invalid_position, table_overflow, APD_locked*	
ApdDelete	(*pos*)	
	fails: *invalid_position, APD_locked*	
ApdGet	()	→ (*cap*[*n_slots*], *n_locked*)
	fails: *protection_violation*	
ApdFlush	()	
	never fails	
ApdLookup	(*adr, mode*)	→ (*adr*)
	fails: *protection_violation*	
ApdLock	(*n_locked*)	
	fails: *invalid_position*	
PdxCall	(*entry_pt, param, pd*) → (*cap*)	
	fails: *protection_violation, invalid_PDX, APD_locked*	

Table 3.1: System calls dealing with protection domains.

ApdInsert: Insert a new Clist into the APD at index *pos*. The slot *pos* must not be locked. The previous contents of slot *pos*, as well as all successive ones, are pushed down (to slot *pos+1* etc). Execute access to the object referenced by *clist_adr* is validated, and, if successful, the capability to the Clist is inserted into the APD. If *pos* is greater than the total number of slots presently in use, the first free slot is used.

ApdDelete: Pop slot *pos*, which must not be locked, i.e., *pos* must be $\geq$ *n_locked*. The following entries are shifted upwards. This removes the Clist capability recoded in slot *pos* from the APD. Due to validation caching, this will not immediately make the Clist's objects inaccessible. However, cached validations are guaranteed to be invalidated after a specific time interval.

ApdFlush: Flush the thread's validation cache immediately and re-validate all Clist capabilities in the APD. This forces all pending revocations to become effective immediately.

ApdGet: Returns the list of Clist pointers and the value of *n_locked*. Only the address part of the APD's Clist capabilities is returned, all passwords are returned as zero.

ApdLookup: Performs an explicit validation of an access of type *mode* to *address*. If successful, returns the address of the first capability granting (at least) the requested rights to the specified address, and caches the validation, otherwise returns NULL. Any previously cached access rights to the object are removed from the validation cache. NULL is also returned if an access is denied by the encounter of a negative capability.

ApdLock: Partially lock the caller's APD by setting its *n_locked* value. The call can only *increase*, not decrease *n_locked*. In other words, it can lock additional slots, but cannot unlock any. All slots with position number *pos*$<$*n_locked* are prevented from modification. If *pos* is *all*, lock the whole APD (i.e. all slots). Any future *ApdInsert*, *ApdDelete* or *PdxCall* operations cannot affect locked slots. *If the whole APD is locked, the thread will no longer be able to perform any ApdInsert, ApdDelete, ObjNewPager, ObjCrePasswd, ObjCrePdx, or PdxCall operations.*

PdxCall: Call a PDX object via entrypoint *adr*, passing *param*. The call will execute in an APD which is the union of the domain passed via the *pd* parameter and the domain registered for the PDX object. The entrypoint called must be contained in the list of entrypoints registered (via *ObjCrePdx*) for the PDX object.

If *pd* is equal to *merge*, the caller's APD is extended by the PDX domain to form the APD of the call. The new Clist is normally inserted in APD slot one (shifting down further slots), similar to a call to *ApdInsert*(1,*adr*), and slot zero is replaced to reference a new system object. Locking slot zero does not prevent its replacement by a reference to a newly created system object. If slot one is locked, the first unlocked slot will be used instead. The system call fails if there are no unlocked slots (the whole APD is locked) or the APD is full. Note that *n_slots=n_pd* does not prevent a *PdxCall*, as this still leaves unlocked slots at the end, unless the APD is full.

If *pd* is equal to *empty*, an empty protection domain is passed to the call, and the call executes just in the protection domain registered for the PDX object (plus the system object, which contains the stack and which is referenced by slot zero). Caller and callee can still share objects if the caller passes a capability explicitly as an argument to the PDX procedure, and the callee then inserts that capability into one of its Clists. (The system object Clist in slot zero is always available for this purpose, and its use is appropriate if the PDX procedure only needs to use the shared object for the duration of the call.)

If *pd* is not one of the above special values, the call executes in a APD constructed from the system object, the Clist registered for the PDX object, and the contents of the PD object. Execute permission is required for the PD object. Unless the caller's whole APD is locked (in which case the *PdxCall* fails), it is irrelevant whether part of the caller's APD is locked.

If the PDX call requires allocation of a new system object (to provide a stack object) this is charged to the caller's default bank account (see Section 4.1).

The PDX procedure returns via a normal function return. Its return value is of type capability, which allows it to return a newly allocated buffer even if caller and callee execute in disjoint protection domains (*pd* equal to *empty*).

If the PDX procedure does not return, but instead exits (by an explicit *ThreadDelete(myself,...)* or a fault), the calling thread is killed. If the PDX procedure creates new threads, and does not kill them prior to returning, these may or may not survive the PDX call. If they survive the return of the PDX call, they may be killed by the system at any time later on.

The thread's exception handlers are reset for the duration of the PDX call. However, the thread can register new handlers during the call, which will remain in force until the call returns (in which case the pre-call settings are once more valid) or until explicitly replaced by new settings. If an exception handler is registered during a PDX call this may effect other threads if they are executing in the same PDX protection domain.

The last-error value is reset as well during the call: *GetLastError* executed at the beginning of the PDX code will return zero. The *PdxCall* syscall can only fail prior to execution of any PDX code, hence its return value indicates whether or not any PDX code was executed. If any PDX code was executed, and the call returns at all, a subsequent call to *GetLastError* will return zero.

Chapter 4

Threads

4.1 Process model

Threads are the basic execution abstraction, they are kernel scheduled. Each thread runs in a protection domain, its *active protection domain* (APD, see Section 3.2). Threads can be created, using the *ThreadCreate* system call, to run in the caller's APD or in a new protection domain, instantiated from a PD object referenced in the *ThreadCreate* call.

Creating a new thread within the caller's APD is a very lightweight operation. The new thread's stack may be supplied by the caller, to reduce the cost of creating threads which need no, or only a very small stack. Otherwise the system allocates a stack from a general stack object shared by several threads.

Creating a thread in a new APD is significantly more heavyweight. A new APD, stack, and environment must be set up. (A stack pointer may be supplied by the caller, but this only works if it references an object which is already in the new thread's protection domain.) The cost can be significantly reduced if the new thread is allowed to *join* other threads running in an APD instantiated from the PD object referenced in the *ThreadCreate* system call. A flag in the *thread_info* parameter indicates whether the new thread may join an existing APD.

When a thread is created in a new APD (either because joining is prohibited by the parameters to the system call, or because there is presently no APD associated with the designated PD object) the system does the following:

- A new "system object" is created. This object is used for:
  - the new thread's stack;
  - the new thread's environment;
  - a Clist containing, among others, the system object's (and thus its own) capability;
  - a "system stack" for upcalls;
  - stacks of any further threads created in the same APD without an explicit stack pointer.

  The Clist set up in that object contains references to some of these items in well known positions. The system object is created with the *persistent* flag off, i.e. it will be cleaned up when the last thread in the new APD exits.

- A new APD is created from a copy of the PD object referenced in the system call. Slot zero of that APD is overwritten with the capability of the Clist set up for the system object. A thread can then use *ApdLookup* to obtain the capability to its own stack.

The bank account to be used for the system object is the new thread's default bank account. This is specified as part of the *thread_info* parameter or, if not given, is inherited from the parent.

The *environment* is essentially a by-value parameter passed to a thread from its parent. Its first word is expected to contain its length, to enable the system to copy it. Otherwise, the environment is completely under user control, the system does not care about its contents. It may be used to store things like bank accounts, address of a directory service, etc., which are typically inherited from the parent. The environment is shared by all threads executing within the same APD.

No heap (data) object is set up for a thread by the kernel. A heap object for a thread can be allocated by the run-time library on demand, and the heap's address can be stored in the environment or the slot-zero Clist.

There is a hierarchy of threads: A thread T_1 created by another thread T_0 cannot survive T_0, unless it is adopted by a thread higher up in the hierarchy. Hence, killing a thread kills the whole hierarchy of threads created by it, unless children are adopted. Only threads in the caller's descendency (i.e. children or more remote offspring) can be killed.

Note: The thread hierarchy as defined at the moment is simplistic and unsatisfactory. It will be revised in a future version.

4.2 System calls

The system interface dealing with threads is given in Table 4.1. The system calls work as follows:

System calls:		
ThreadCreate	(*entry_pt, param, info, pd*)	→ (*thread_id*)
	fails: *protection_violation, table_overflow, invalid_info, invalid_Clist*	
ThreadDelete	(*thread_id, status, adopt*)	
	fails: *protection_violation, invalid_thread*	
ThreadSleep	(*thread_id, time*)	
	fails: *protection_violation, invalid_thread*	
ThreadResume	(*thread_id*)	
	fails: *protection_violation, invalid_thread*	
ThreadWait	(*thread_id*)	→ (*thread_id, status*)
	fails: *protection_violation, invalid_thread*	
ThreadMyId	()	→ (*thread_id*)
	never fails	
ThreadInfo	(*thread_id, thread_info*)	→ (*thread_info*)
	fails: *protection_violation, invalid_thread*	

Table 4.1: System calls dealing with threads.

ThreadCreate: Creates a new thread which starts to execute at *entry_pt* and has arguments *param*. The new thread is to execute in a protection domain instantiated from *pd*. If *pd* is null, the new thread will share the caller's APD. If *pd* is supplied it must be a protection domain special object, and the caller must hold execute permission to it. The *info* argument specifies further thread attributes, such as time and memory limits and stack size, the caller must have read access to it. A stack pointer may be specified in *info* which is then used for the new thread. If no stack pointer value is supplied by the caller, the system will allocate a stack from the APD's system object, unless *info* specifies a stack size of zero,

in which case the thread is started with an invalid stack pointer. The *detached* flag indicates whether or not the thread is cleaned up immediately when killed, see *ThreadDelete* below.

If the *join* flag is set and *pd* is non-null, the thread can be started in an existing APD, if any, which was instantiated from the specified PD object. In this case the new thread's APD may later be joined by other threads started with the same *pd* parameter and the *join* flag set. Note that if a thread modifies its APD, this will affect all threads sharing that APD. Starting a thread without the *join* flag may be more expensive but ensures that the thread does not share its APD with any other threads, unless it later performs an explicit *ThreadCreate* system call with a null *pd* parameter.

If a new system object is allocated during the call (*pd* was non-null and no APD could be joined) the *bank_account* supplied in *info* is used for the new object. The call will fail in this case if no valid bank account was supplied.

A new environment is only created when a new system object is created. The thread (and thus its APD) receives a null environment if *info.env_size* is zero. Otherwise the new environment is created by copying the string pointed to by *info.environment*. If this pointer is null, the caller's environment is copied instead. Note that the first word of the environment contains the actual length (in words). The amount copied is the lesser of that length, and *info.env_size*.

ThreadDelete: Kills the specified thread, which must be a descendent of the caller. The *status* value will be returned to a thread waiting for the target thread to exit. A *thread_id* of *myself* kills the caller. If *adopt* is TRUE, any children of the target thread will be adopted by the caller, otherwise all direct or indirect descendents will be killed. A thread created with the *detached* attribute set in *thread_info* will be cleaned up immediately when killed, otherwise cleanup is deferred until the parent has performed a *ThreadWait* to collect the thread's exit status (or the parent is cleaned up).

ThreadSleep: Stops a thread, which must be a descendent of the caller, for a specified interval of real time, or until explicitly awaken by *ThreadResume*. A *thread_id* of *myself* stops the caller. A *time* value of *forever* blocks the thread indefinitively, until an explicit *ThreadResume*. A thread stopping itself for a zero time interval performs a *yield* operation, i.e. forfeits the remainder of its time slice.

time is specified in nano-seconds. The actual resolution will be coarser.

ThreadResume: Resume a sleeping thread, which must be a descendent of the caller.

ThreadWait: Wait until the specified thread, which must be a *direct* descendent of the caller, is killed. If *thread_id* is *any*, wait for any thread to be killed. Returns the *thread_id* of the thread that was killed, plus the status supplied when killing it. The call will return immediately, returning a *thread_id* of *any* and an undefined *status*, if there is nothing to wait for (i.e. waiting for a non-existent thread is not considered an error).

ThreadMyId: Return the ID of the calling thread.

type *ThreadInfo* $\equiv$ {
stack, *stack_size*, *environment*, *env_size*,
mem_limit, *time_limit*, *cpu_time*, *priority*, *start_time*,
flags, *bank_account*
}

Figure 4.1: Thread descriptor data structure.

ThreadInfo: Sets attributes of the specified thread, which must a descendent of the caller; *myself* means the calling thread. Zero values for *prio*, *cpu_lim*, *mem_lim*, *thread_info* mean no limit. Limits can only be restricted, not relaxed by this call. Returns pre-call values of thread attributes, in the format shown in Figure 4.1. Attempts to modify *stack*, *stack_size*, *start_time*, *sys_time* and *user_time* are ignored.

Chapter 5

Page Fault Handlers and Virtual Memory Mappings

5.1 User-level page fault handlers

When an object is created, no backing store or physical memory is allocated initially. Hence, an access to any of the object's pages will lead to a page fault. Initially, such page faults are handled by the system's *default pager*, which will allocate a disk block for the accessed page, allocate a zero-filled physical frame, and enter the appropriate information into the faulting thread's page table. Page replacement and residency faults are handled in the usual fashion.

Alternatively, threads can register their own page fault handlers for particular objects, using the *ObjNewPager* call (see Table 2.1). This system call is passed a pointer to a PDX entrypoint; a null pointer re-instates the default pager.

An object's pager is called by the kernel on behalf of the faulter (using the faulter's thread ID) whenever a page fault happens on the object. It is invoked by a *PdxCall* passing an empty protection domain (c.f. Section 3.5). Hence the pager executes in just the protection domain defined for it in its OT entry.

There are three kinds of page faults:

residency fault: an access failed because the page was not resident, the pager should establish a mapping for the faulting page (or indicate failure);

write fault: a write access was attempted on a read-only (R/O) page, the pager should establish a R/W mapping to the page (or indicate failure);

flush: a *PageFlush* operation was requested for a page; the pager should ensure that the page is clean (usually by forwarding the flush operation to the object the faulting object is mapped to).

5.2 Virtual memory mapping operations

The Mungi kernel interface does not provide for any explicit I/O operations. Instead, all devices are memory-mapped. The default pager performs I/O between physical memory and paging disk, and establishes mappings between virtual pages and physical frames.

User page fault handers, since they are normal user code, have no access to physical devices. Instead they use virtual memory operations to map one virtual memory (VM) region (the object whose page faults are handled by the pager) to another, which is handled by another pager. Eventually, the mapping chain must end at an object handled by the default pager.

Virtual memory mappings introduce *aliasing* — the same data (physical frame) is potentially visible at different virtual addresses. However, using aliases for accessing data is *strongly discouraged*: Mungi makes **absolutely no guarantee** about any consistency between data accessed via aliases. All accesses to data should *always* use the same virtual memory address.

5.3 System calls

Table 5.1 lists system calls dealing with virtual memory mappings. The meaning of the calls is as follows:

System calls:		
PageCopy	(*from_page, to_page, n_pages*)	
	fails: *protection_violation, invalid_range, invalid_null_value*	
PageMap	(*from_page, to_page, n_pages, mode, fault_in*)	
	fails: *protection_violation, invalid_range, invalid_null_value*	
PageUnmap	(*page, n_pages, disp*)	
	$disp \in \{zero, replace, unalias, keep\}$	
	fails: *protection_violation, invalid_range, invalid_null_value*	
PageFlush	(*page, n_pages*)	
	fails: *protection_violation, invalid_range, invalid_null_value*	
Pager signature:		
Pager	(*page, n_pages, fault_type*)	$\rightarrow$ (*success* $\|\|$ *fail*)
	$fault\_type \in \{miss, write, flush\}$	

Table 5.1: System calls dealing with virtual memory mappings and call interface for user-level pagers.

PageCopy: Copy a range of pages, using copy-on-write (where possible). The two ranges must not overlap, and each range must be completely contained within an object. Requires R capability on the source and W capability on the destination.

PageMap: Alias two page ranges, each of which must be fully contained within an object. If *mode* is *read_only*, a subsequent write attempt on the range starting with *to_page* will result in a write fault. The two ranges must not overlap, unless they are identical and *mode* is *read_only*, in which case the call serves to turn on write protection on the range of pages. Unless the ranges are identical, this operation implies *PageUnmap(to_page,n_pages,zero)*.

The parameter *fault_in* determines what happens if some source pages are not resident. If *fault_in* is TRUE, such pages are forced to become resident (by invoking their pager). Otherwise, *PageMap* is a no-op where the source pages are not resident.

The operation requires RW capability on the destination. On the source, R capability is required if *mode* is *read_only*, otherwise RW capability is required.

The alias so established between two virtual pages vanishes as soon as either the source or destination page becomes non-resident for whatever reason (VM page replacement or explicit unmap). Therefore this operation is only of use for page fault handlers.

PageUnmap: Invalidate page mappings. The whole range of pages must be part of a single object. If *disp* is *unalias*, any mappings **from** the specified pages are removed, i.e. all destination pages which used

the specified pages as the source of their mappings are unmapped. For objects handled by a user-level pager, all other values of *disp* lead to the pages being simply unmapped. For objects handled by the default pager, the semantics depend on the *disp* parameter. If *disp* is *zero*, the pages loose all association with physical memory or backing store, reverting them to the state they were in when the object was originally created by an *ObjCreate* call. If *disp* is *replace*, the virtual pages' association with any physical frames is lost, but any association with backing store is retained. However, dirty pages are **not** flushed to backing store, an explicit *PageFlush* call needs to be performed first if this is desired. If *disp* is *keep*, the pages' mappings are not lost at all, but are marked as invalid, forcing pager invocation on the next access. This can be used to force updating of time stamps on the next access.

RW capability is required, except for R/O mapped pages handled by a user-level pager, where R capability is sufficient.

PageFlush: Clean a range of pages, which must be part of a single object. For pages handled by the default pager, ensures that any modified pages are flushed to disk, **and** that their association with backing store is recorded in stable storage. For pages handled by a user-level pager, this is simply translated into a pager invocation with a *fault_type* of *flush*. The operation is a no-op on clean or non-resident pages. Requires RW capability.

Pager: The calling convention of a user-level pager.

Chapter 6

Miscellaneous System Calls

6.1 System call error codes

System calls return a value indicating whether the call was successful or not. If not the *GetLastError()* syscall can be used to retrieve the error code. The value returned by *GetLastError()* is the error code of the last syscall performed by the same thread. If the thread has not performed any prior syscalls, or if the previous syscall was successful, a value of zero is returned.

6.2 Exceptions

Exceptions may be generated as a result of a program fault (e.g. division by zero or protection fault). An exception handler can be registered to handle an event. If an exception occurs for which a handler had been registered, that handler is called as an un-programmed function call of the thread which caused the exception. To handle an exception which is not associated with a particular thread, the system selects any thread within the faulting APD.

Exception handlers are associated with an APD. New threads created in an existing APD (via *ThreadCreate* with a null *pd* parameter or by joining) have the same exception handlers as other threads in that APD. Any thread setting or changing an exception handler will affect all other threads belonging to the same APD.

A thread started in a new APD has no exception handlers associated with it. The same applies to an APD created during a *PdxCall*, the thread executing the PDX procedure has no exception handlers during the time of the execution, unless it (or another thread in the same PDX APD) registers one by calling *ExcptReg*.

6.3 Semaphores

Semaphores are used for synchronisation. A semaphore is identified by a byte in an object. The association of a semaphore with a particular address serves to name semaphores and to integrate them with Mungi's protection system. The contents of the byte addressed by the semaphore's name has nothing to do with the state of the semaphore, and it can be used independently of the semaphore.

6.4 System calls

The system calls are listed in Table 6.1 and explained below.

GetLastError: Returns the error status of the last system call made by the calling thread.

System call errors:	
GetLastError ()	→ (*error_no*)
never fails	

Asynchronous exceptions:	
ExcptReg	(*exception, handler_adr*) → (*old_handler*)
	fails: *inv_exception*

Exception handler signature:	
handler	(*exception, address*)

Semaphores:	
SemCreate	(*address, value, flags*)
	fails: *protection_violation, in_use, too_many_semaphore*
SemDelete	(*address*)
	fails: *protection_violation, inv_semaphore*
SemWait	(*address*)
	fails: *protection_violation, inv_semaphore, semaphore_deleted*
SemSignal	(*address*)
	fails: *protection_violation, inv_semaphore*

Table 6.1: Miscellaneous system calls.

ExcptReg: Register a handler for *exception*. Returns the address of the previously registered handler. NULL indicates no handler, i.e., the system will take a default action if the exception arises (usually the default action is to terminate the faulting thread).

handler: Calling convention for an exception handler. A handler executes on the faulting thread's stack. Handlers return to the point where the exception happened, or can use the *setjmp, longjmp* facility. If the handler is killed, this will kill the faulting thread.

SemCreate: Create a semaphore named by the specified *address* and initialised to *value*. Requires R/W capability on the object containing *address*.

The *flags* modify the semaphore semantics, in particular, the order in which waiting threads get awakened when the semaphore is signalled. Default is the standard (fair) semaphore behaviour: waiting threads are awakened in FIFO order. The flags change this as follows:

wake_lifo: The last thread that did a *SemWait* on the semaphore will be awakened when the semaphore is signalled.

wake_affinity: Modifies FIFO or LIFO behaviour: Threads whose affinity are to the same CPU as the signalling thread are woken in priority over remote threads.

wake_all: All waiting threads are awakened when the semaphore is signalled. Obviously this is no longer proper semaphore behaviour and unsuitable for mutual exclusion. This flag must not be specified together with any of the other flags.

SemDelete: Delete a semaphore. Any threads waiting on the semaphore will be faulted. Requires R/W capability on the object containing *address*.

SemWait: Perform a *wait* operation on the specified semaphore, i.e. do atomically:

$$\underline{\textbf{while}}\ sem \leq 0\ \underline{\textbf{do}}\ \ \underline{\textbf{od}};$$
$$sem := sem - 1;$$

The implementation does not use a busy wait. Requires R capability on the object containing *address*.

SemSignal: Perform a *signal* operation on the specified semaphore, i.e. do atomically:

$$sem := sem + 1;$$

Requires R capability on the object containing *address*.

Bibliography

[APW86] M. Anderson, Ronald Pose, and Chris S. Wallace. A password-capability system. *The Computer Journal*, 29:1–8, 1986. 2

[Ber80] Viktors Berstis. Security and protection in the IBM System/38. In *Proceedings of the 7th Symposium on Computer Architecture*, pages 245–250. ACM/IEEE, May 1980. 1, 11

[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 *Proceedings of the 3rd Workshop on Workstation Operating Systems*, pages 80–85, Key Biscayne, FL, USA, 1992. IEEE. 1

[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 Transactions on Computer Systems*, 12:271–307, 1994. 1

[DH99] Luke Deller and Gernot Heiser. Linking programs in a single address space. In *Proceedings of the 1999 USENIX Technical Conference*, pages 283–294, Monterey, Ca, USA, June 1999. 1

[ERHL96] Kevin Elphinstone, Stephen Russell, Gernot Heiser, and Jochen Liedtke. Supporting persistent object systems in a single address space. In *Proceedings of the 7th International Workshop on Persistent Object Systems (POS)*, pages 111–119, Cape May, NJ, USA, May 1996. Morgan Kaufmann. 1

[HERH93] Gernot Heiser, Kevin Elphinstone, Stephen Russell, and Graham R. Hellestrand. A distributed single address space system supporting persistence. Technical Report UNSW-CSE-TR-9302, University of NSW, University of NSW, Sydney 2052, Australia, March 1993. 1

[HERV94] Gernot Heiser, Kevin Elphinstone, Stephen Russell, and Jerry Vochteloo. Mungi: A distributed single-address-space operating system. In *Proceedings of the 17th Australasian Computer Science Conference (ACSC)*, pages 271–80, Christchurch, New Zealand, January 1994. 1

[HEV+98] Gernot Heiser, Kevin Elphinstone, Jerry Vochteloo, Stephen Russell, and Jochen Liedtke. The Mungi single-address-space operating system. *Software: Practice and Experience*, 28(9):901–928, July 1998. 1

[HLR98] Gernot Heiser, Fondy Lam, and Stephen Russell. Resource management in the Mungi single-address-space operating system. In *Proceedings of the 21st Australasian Computer Science Conference (ACSC)*, pages 417–428, Perth, Australia, February 1998. Springer-Verlag. Also available as UNSW-CSE-TR-9705 from http://www.cse.unsw.edu.au/school/research/tr.html. 3, 4

[HSH81] Merle E. Houdek, Frank G. Soltis, and Roy L. Hoffman. IBM System/38 support for capability-based addressing. In *Proceedings of the 8th Symposium on Computer Architecture*, pages 341–348. ACM/IEEE, May 1981. 1

[Lie95] Jochen Liedtke. On μ-kernel construction. In *Proceedings of the 15th ACM Symposium on OS Principles (SOSP)*, pages 237–250, Copper Mountain, CO, USA, December 1995. 1

[RSE+92] Stephen Russell, Alan Skea, Kevin Elphinstone, Gernot Heiser, Keith Burston, Ian Gorton, and Graham Hellestrand. Distribution + persistence = global virtual memory. In *Proceedings of the 2nd IEEE International Workshop on Object Orientation in Operating Systems (IWOOOS)*, pages 96–99, Dourdan, France, September 1992. 1

[Sol96] Frank G. Soltis. *Inside the AS/400*. Duke Press, Loveland, CO, USA, 1996. 1

[VERH96] Jerry Vochteloo, Kevin Elphinstone, Stephen Russell, and Gernot Heiser. Protection domain extensions in Mungi. In *Proceedings of the 5th IEEE International Workshop on Object Orientation in Operating Systems (IWOOOS)*, pages 161–165, Seattle, WA, USA, October 1996. 1

[Voc98] Jerry Vochteloo. *Design, Implementation and Performance of Protection in the Mungi Single-Address-Space Operating System*. Phd thesis, School of Computer Science and Engineering, University of NSW, Sydney 2052, Australia, July 1998. Available from `http://www.cse.unsw.edu.au/~disy/papers/`. 12

[VRH93] Jerry Vochteloo, Stephen Russell, and Gernot Heiser. Capability-based protection in the Mungi operating system. In *Proceedings of the 3rd IEEE International Workshop on Object Orientation in Operating Systems (IWOOOS)*, pages 108–15, Asheville, NC, USA, December 1993. 1

[WM96] Tim Wilkinson and Kevin Murray. Evaluation of a distributed single address space operating system. In *Proceedings of the 16th IEEE International Conference on Distributed Computing Systems (ICDCS)*, pages 494–501, Hong Kong, May 1996. IEEE. 1

[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 of NSW, University of NSW, Sydney 2052, Australia, November 1995. 1

[WSO+92] Tim Wilkinson, Tom Stiemerling, Peter E. Osmon, Ashley Saulsbury, and Paul Kelly. Angel: A proposed multiprocessor operating system kernel. In *European Workshop on Parallel Computing*, pages 316–319, Barcelona, Spain, 1992. 1

Appendix A

C Language Bindings

A.1 include/sys/types.h

```
/*****************************************************************************
 *
 *      $Id: types.h,v 1.16.2.1 2002/08/29 04:31:54 cgray Exp $
 *      Copyright (C) 2002 Operating Systems Research Group, UNSW, Australia.
 *
 *      This file is part of the Mungi operating system distribution.
 *
 *      This program is free software; you can redistribute it and/or
 *      modify it under the terms of the GNU General Public License
 *      version 2 as published by the Free Software Foundation.
 *      A copy of this license is included in the top level directory of
 *      the Mungi distribution.
 *
 *      This program is distributed in the hope that it will be useful,
 *      but WITHOUT ANY WARRANTY; without even the implied warranty of
 *      MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *      GNU General Public License for more details.
 *
 *****************************************************************************/

#ifndef __MUNGI_TYPES_H
#define __MUNGI_TYPES_H

#include <types.h>

#ifndef NULL
#define NULL 0
#endif

/* Types */
#if defined(MIPSENV)
#if defined(_MIPS_SZPTR) && (_MIPS_SZPTR == 64)
#else
#error Need 64 bit compiler to build Mungi - sorry!
#endif
#if defined(__GNUC__) && __GNUC__ < 3
```

```
typedef unsigned char           _Bool;  /* part of new C Standard */
#endif
typedef signed char             int8_t;
typedef unsigned char           uint8_t;
typedef short                   int16_t;
typedef unsigned short          uint16_t;
typedef int                     int32_t;
typedef unsigned int            uint32_t;
typedef long                    int64_t;
typedef unsigned long           uint64_t;
#ifndef _SIZE_T
#define _SIZE_T
typedef uint64_t                size_t;
#endif
typedef int64_t                 ssize_t;
typedef uint64_t                uintptr_t;

#elif defined(ALPHAENV)
#if defined(__GNUC__) && __GNUC__ < 3
typedef unsigned char           _Bool;  /* part of new C Standard */
#endif
typedef signed char             int8_t;
typedef unsigned char           uint8_t;
typedef short                   int16_t;
typedef unsigned short          uint16_t;
typedef int                     int32_t;
typedef unsigned int            uint32_t;
typedef long                    int64_t;
typedef unsigned long           uint64_t;
#ifndef _SIZE_T
#define _SIZE_T
typedef uint64_t                size_t;
#endif
typedef int64_t                 ssize_t;
typedef uint64_t                uintptr_t;

#else /*  #if defined(arch) */
#error Mungi not ported to this architecture yet!!
#endif /* #if defined(arch) */

#define bool _Bool
typedef unsigned int            uint;
typedef int8_t                  apdpos_t;
typedef uint64_t                passwd_t;

/* capability */
typedef struct {
        void *address;
        passwd_t passwd;
} cap_t;

/*
 * Access Rights stuff
```

```
 */
typedef int8_t access_t;
typedef int8_t mac_access_t;

#define M_EXECUTE       ((access_t) 1<<0)
#define M_WRITE         ((access_t) 1<<1)
#define M_READ          ((access_t) 1<<2)
#define M_DESTROY       ((access_t) 1<<3)
#define M_PDX           ((access_t) 1<<4)
#define M_NOT           ((access_t) 1<<5)
#define M_SYNC          (M_READ|M_WRITE)
#define M_OWNER         (M_EXECUTE|M_READ|M_WRITE|M_DESTROY|M_SYNC)

/* MAC validation request sent to policy object and returned with the result */
struct validation_request {
        unsigned long subject_label:60;
        unsigned long set_0_or_1:1;
        unsigned long read_or_create_domain:1;
        unsigned long write_or_create_type:1;
        unsigned long execute_or_transfer:1;
        unsigned long object_label:60;
        unsigned long destroy_or_pdx:1;
        unsigned long result:1;
        unsigned long undef:2;
};


/************\
 *  Objects *
\************/

/* time */
typedef uint64_t time_t;
#define SLEEP_INFINITY  ((uint64_t)-1)


/* Flags specifying what to modify with an ObjectInfo call */

#define O_SET_NONE      0x00  /* don't change */
#define O_SET_MODIFY    0x01  /* set modify time */
#define O_SET_ACCESS    0x02  /* set access time */
#define O_SET_ACCNT     0x04  /* set accounting time */
#define O_SET_TYPE      0x08  /* set the security type label */
#define O_SET_MASK      0x0f

#define O_TCH_MODIFY    0x10  /* touch modify time */
#define O_TCH_ACCESS    0x20  /* touch access time */
#define O_TCH_ACCNT     0x40  /* touch accounting time */
#define O_TCH_MASK      0xf0


/* general object flags */
```

```
typedef int objflags_t;
#define O_PERS          ((objflags_t)0x001)     /* is persistent */

/* special object flags */
#define O_PD            ((objflags_t)0x001)     /* A PD object */
#define O_ACCT          ((objflags_t)0x002)     /* is a bank account */
#define O_ACCT_F        ((objflags_t)0x004)     /* bank account is financial */


#define O_MAX_CAPS      0x80
#define O_MAX_PDX       0x10
#define O_MAX_ENTPT     0xC0


#define PD_MERGE        (apddesc_t *)-1
#define PD_EMPTY        (apddesc_t *)0

typedef cap_t (*pdx_t)(cap_t); /* PDX function with arbitrary arg */
typedef struct {
        cap_t      clist[O_MAX_PDX];
        passwd_t  passwd[O_MAX_PDX];
        int       n_entry[O_MAX_PDX];
        int       x_entry[O_MAX_PDX];
        pdx_t    entry[O_MAX_ENTPT];
} pdxdata_t;

/* public object info */
typedef struct {
        /* public */
        size_t          extent;   /* block size to use on backing store */
        time_t          creation;
        time_t          modification;
        time_t          access;
        time_t          accounting;
        void            *userinfo;
        void            *acctinfo;
        size_t          length;

        /* private */
        objflags_t      flags;
        objflags_t      special;
        uint            n_caps;
        uint            n_pdx;
        cap_t           account;
        cap_t           pager;
        cap_t           cntrl_object;
        passwd_t        passwd[O_MAX_CAPS]; /* separate password & rights for */
        access_t        rights[O_MAX_CAPS]; /* smaller space due to alignment */
        pdxdata_t       pdx;
} objinfo_t;
```

```
/*******\
 * APD *
\*******/

#define APD_MAX_ENTRY      0x10
typedef struct {
        cap_t           clist[APD_MAX_ENTRY];   /* address of Clists */
        apdpos_t        n_locked;               /* slot number that is locked */
        apdpos_t        n_apd;                  /* number of slots in use */
} apddesc_t;


typedef uint8_t clistformat_t;
#define CL_UNSRT_0      ((clistformat_t)0x1)    /* unsorted format */
#define CL_SRT_0        ((clistformat_t)0x2)    /* sorted format */

typedef struct {
        char            type;           /* magic number 'c' */
        uint8_t         rel_ver;        /* Presently 1 */
        clistformat_t   format;
        uint16_t        n_caps;         /* size of Clist */
        uint32_t        reserved;
        cap_t           caps[1];        /* cap array of size n_caps */
} clist_t;




/**********\
 * Threads *
\**********/

typedef int (*thread_t)(void *); /* thread function with arbitrary arg */

typedef uint64_t         mthreadid_t;
#define THREAD_MYSELF   ((mthreadid_t)0)
#define THREAD_ANY      ((mthreadid_t)0)
#define THREAD_NULL     ((mthreadid_t)0)


/* flags used to specify what parameters we supplied */
#define THREAD_STACK_ADDR       (1 << 0) /* a stack address was specified */
#define THREAD_STACK_SIZE       (1 << 1) /* a stack size was specified */
#define THREAD_MEM_LIMIT        (1 << 2) /* memory limit was specified */
#define THREAD_CPU_TIME         (1 << 3) /* time limit was specified */
#define THREAD_DETACHED         (1 << 4) /* start the thread detached (ie dont
                                            wait for it to finish */
#define THREAD_BANK_ACCOUNT     (1 << 5)
#define THREAD_NO_JOINPD        (1 << 6) /* instantiate from PD */

typedef struct {
        int32_t flags; /* flags indicating thread parameters filled in*/
        uint    prio;
```

```
        void    *stack_addr;
        size_t  stack_size;
        time_t  start_time;
        time_t  cpu_time;
        time_t  cpu_limit;
        size_t  mem_limit;
        void    *bank_account;
        void    *env;   /* environment */
        size_t  env_size;
} threadinfo_t;


/*****************\
 * Page Mappings *
\*****************/

typedef int pagefault_t;
#define PF_RESID        ((pagefault_t)1)        /* Residency fault */
#define PF_WRITE        ((pagefault_t)2)        /* Write fault */
#define PF_FLUSH        ((pagefault_t)3)        /* Flush event */

typedef int pagedisp_t;
#define P_DSP_ZERO      ((pagedisp_t)1)
#define P_DSP_REPLACE   ((pagedisp_t)2)
#define P_DSP_KEEP      ((pagedisp_t)3)
#define P_DSP_UNALIAS   ((pagedisp_t)4)

typedef bool (*pager_t)(const void *, size_t n_pages, pagefault_t);

#endif  /* __MUNGI_TYPES_H */
```

A.2 include/status.h

```
/****************************************************************************
 *
 *      $Id: status.h,v 1.9 2002/08/23 08:24:15 cgray Exp $
 *      Copyright (C) 2002 Operating Systems Research Group, UNSW, Australia.
 *
 *      This file is part of the Mungi operating system distribution.
 *
 *      This program is free software; you can redistribute it and/or
 *      modify it under the terms of the GNU General Public License
 *      version 2 as published by the Free Software Foundation.
 *      A copy of this license is included in the top level directory of
 *      the Mungi distribution.
 *
 *      This program is distributed in the hope that it will be useful,
 *      but WITHOUT ANY WARRANTY; without even the implied warranty of
 *      MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *      GNU General Public License for more details.
 *
 ****************************************************************************/

/*
 *    Mungi status values
 */

#ifndef __MUNGI_STATUS_H_
#define __MUNGI_STATUS_H_

#define ST_SUCC         0x00    /* successful */
#define ST_NOMEM        0x01    /* out of memory */
#define ST_SIZ          0x02    /* invalid size */
#define ST_POS          0x04    /* invalid position */
#define ST_CAP          0x05    /* invalid capability */
#define ST_CLIST        0x06    /* invalid C-list */
#define ST_PWD          0x07    /* invalid password */
#define ST_INFO         0x08    /* invalid information */
#define ST_NULL         0x09    /* invalid NULL value */
#define ST_LOCK         0x11    /* APD locked */
#define ST_NOGROW       0x12    /* cannot grow */
#define ST_OVFL         0x13    /* table overflow */
#define ST_THR          0x14    /* invalid thread ID */
#define ST_PROT         0x16    /* protection violation */
#define ST_RNG          0x17    /* invalid range */
#define ST_EXCPT        0x18    /* invalid exception */
#define ST_USE          0x19    /* semaphore in use */
#define ST_SEMA         0x1a    /* invalid semaphore */
#define ST_NOIMP        0x1b    /* syscall not implemnted */
#define ST_ERR          0x1c    /* something really bad happened */
#define ST_SEMLMT       0x1d    /* no room for more semaphores */
#define ST_SDEL         0x1e    /* the semaphore has been deleted while we
                                        were waiting on it */
#define ST_BANK         0x1f    /* invalid bank account data */
```

```
#define ST_PDX          0x20    /* invalid pdx data */

#endif /* __MUNGI_STATUS_H_ */
```

A.3 include/exception.h

```
/*****************************************************************************
 *
 *      $Id: exception.h,v 1.3 2002/05/31 05:20:11 danielp Exp $
 *      Copyright (C) 2002 Operating Systems Research Group, UNSW, Australia.
 *
 *      This file is part of the Mungi operating system distribution.
 *
 *      This program is free software; you can redistribute it and/or
 *      modify it under the terms of the GNU General Public License
 *      version 2 as published by the Free Software Foundation.
 *      A copy of this license is included in the top level directory of
 *      the Mungi distribution.
 *
 *      This program is distributed in the hope that it will be useful,
 *      but WITHOUT ANY WARRANTY; without even the implied warranty of
 *      MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *      GNU General Public License for more details.
 *
 *****************************************************************************/

#ifndef __MUNGI_EXCEPTION_H
#define __MUNGI_EXCEPTION_H


#define EXP_MAX 16 /* Maximum number of exceptions */

typedef unsigned int excpt_t;

typedef void (*excpthndlr_t)(excpt_t, void *);

/**************\
 * Exceptions *
\**************/
#define E_KILL  ((excpt_t)1)    /* thread killed */
#define E_PROT  ((excpt_t)2)    /* protection violation */
#define E_ARITH ((excpt_t)3)    /* arithmetic exception */
#define E_UPCL  ((excpt_t)4)    /* upcall failed */
#define E_ILL   ((excpt_t)5)    /* illegal instruction */

#endif /* __MUNGI_EXCEPTION_H */
```

A.4 include/syscalls.h

```
/******************************************************************************
 *
 *      $Id: syscalls.h,v 1.17.2.1 2002/08/29 04:31:53 cgray Exp $
 *      Copyright (C) 2002 Operating Systems Research Group, UNSW, Australia.
 *
 *      This file is part of the Mungi operating system distribution.
 *
 *      This program is free software; you can redistribute it and/or
 *      modify it under the terms of the GNU General Public License
 *      version 2 as published by the Free Software Foundation.
 *      A copy of this license is included in the top level directory of
 *      the Mungi distribution.
 *
 *      This program is distributed in the hope that it will be useful,
 *      but WITHOUT ANY WARRANTY; without even the implied warranty of
 *      MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *      GNU General Public License for more details.
 *
 ******************************************************************************/

/* Mungi system calls */

#ifndef __MUNGI_SYSCALLS_H
#define __MUNGI_SYSCALLS_H

#include <sys/types.h>
#include <exception.h>


/***********\
 * Objects *
\***********/


void *
ObjCreate ( size_t size, passwd_t passwd, const objinfo_t * info );
/*
 * Allocates object of "size" bytes with owner password "passwd" and object
 * info "*info".
 *
 * Return: object address if successful, NULL otherwise.
 * errors: ST_NOMEM, ST_INFO, ST_BANK.
 */

int
ObjDelete ( void * obj );
/*
 * Deallocates object which contains address "obj".
 *
 * Return: zero if successful, otherwise !=0
 * errors: ST_PROT.
```

```
 */

int
ObjResize ( void * obj, size_t new_size );
/*
 * Resizes object which contains address "obj" to new size "new_size".
 * Call will fail if object cannot be extended in situ.
 *
 * Return: zero if successful, otherwise !=0
 * errors: ST_PROT, ST_INFO, ST_NOGROW, ST_NOMEM.
 */

int
ObjPasswd ( cap_t cap, access_t mode );
/*
 * Registers a new capability conferring rights "mode".
 * or if mode is zero, deletes an existing capability matching cap.passwd.
 *
 * Return: zero if successful, otherwise !=0
 * errors: ST_PROT, ST_OVFL.
 */

int
ObjInfo ( const void * obj, int flags, objinfo_t * info /* INOUT */ );
/*
 * Update the object table entry for "obj". The parameter "flags" specifies
 * which fields are changed.
 * Returns, through "info", the pre-call attribute settings.
 *
 * Return: zero if successful, otherwise !=0
 * errors: ST_PROT, ST_INFO.
 */

int
ObjCrePdx(cap_t cap, const clist_t * clist, uintptr_t unused1,
          uintptr_t unused2, uintptr_t unused3, uint n_entrypt,
          pdx_t entry_pnts[]);
/*
 * Registers a new PDX capability cap.
 * When called, the PDX procedure will extend the callers protection
 * domain by the specified "clist". PDX calls via the new capability are valid
 * to one of the "n_entrypt" specified entry points in entry_pnts[].
 *
 * Return: zero if successful, otherwise !=0
 * errors: ST_PROT, ST_OVFL, ST_NULL, ST_CLIST, ST_PDX, ST_LOCK.
 */

int
ObjNewPager ( void * obj, pager_t pager );
/*
 * Registers PDX function "pager" as the page fault handler for "obj".
 *
 * Return: zero if successful, otherwise !=0
```

```
 * errors: ST_PROT, ST_PDX, ST_LOCK.
 */



/*******\
 * APD *
\*******/

int
ApdInsert ( apdpos_t pos, const clist_t * clist );
/*
 * Insert at position "pos" in the kernel's APD data structure a capability
 * for object "clist". The slot "pos" must not be locked. Slot "pos" and all
 * further slots are shifted downwards.
 *
 * Return: zero if successful, otherwise !=0
 * errors: ST_PROT, ST_POS, ST_OVFL, ST_LOCK.
 */

int
ApdDelete ( apdpos_t pos );
/*
 * Pop slot "pos" (which must not be locked) from the kernel's APD data
 * structure (shifting up any further entries).
 *
 * Return: zero if successful, otherwise !=0
 * errors: ST_POS, ST_LOCK.
 */

int
ApdGet ( apddesc_t * apd /* OUT */);
/*
 * Return a copy of the kernel's APD data structure through "apd". Only the
 * address part of Clist caps are returned (no passwords).
 * "apd->n_locked" returns the locking status for the APD.
 * (slot i is locked iff (i <= n_locked) ).
 *
 * Return: zero if successful, otherwise !=0
 * errors: ST_PROT.
 */

int
ApdLock ( apdpos_t pos );
/*
 * Lock the slot pos in the caller's APD, imposing restrictions
 * on a number of system calls.
 * A slot value of -1 locks the whole APD.
 * A locked APD (slot) cannot be unlocked.
 * Locking an already locked APD (slot) has no effect.
 * All slots below pos are also locked.
 *
 * Return: zero if successful, otherwise !=0
```

```
 * errors: ST_POS.
 */

cap_t *
ApdLookup ( const void * address, access_t minrights );
/*
 * Validate access to "address" of type "minrights".
 * If the access is allowed, the validation is cached as a side effect.
 *
 * Return: NULL if this access is not possible, otherwise the address
 *              (within a C-list) of the capability granting at least the
 *              requested rights.
 * errors: ST_PROT.
 */

void
ApdFlush ( void );
/*
 * Flush the validation cache, forcing revalidation to occur for all future
 * object accesses. Also perform re-validation of Clist caps in the APD.
 */

int
PdxCall ( pdx_t proc, cap_t param, cap_t *ret /* OUT */, const apddesc_t *pd );
/*
 * Call "proc" as a PDX procedure.
 *
 * If "pd" is PD_MERGE, the PDX procedure executes in a protection domain
 * which is the union of the caller's APD with the Clist registered for
 * the PDX.
 * If "pd" is PD_EMPTY, no protection domain is passed, and the PDX executes in
 * an APD consisting solely of its registered Clist. Otherwise, the extended
 * APD is constructed from the registered Clist and the protection
 * domain explicitly passed via pd (a pointer to a PD object).
 * "param" specifies an arbitrary parameter passed to the PDX by value.
 * "ret" is the return value from the PDX call.
 *
 * Return: zero if successful, otherwise !=0
 * errors: ST_PROT, ST_PDX, ST_LOCK.
 */



/**********\
 * Threads *
\**********/

mthreadid_t
ThreadCreate(thread_t ip, void *param, const threadinfo_t *info,
                const apddesc_t *pd);
/*
 * Start a new thread starting execution at address "ip" with parameter "param"
 * "info" specifies optional attributes for the thread creation. A value of
```

```
 * NULL will use default thread attributes.
 * "pd" specifies optional APD information. If pd is NULL the newthread shares
 * the callers APD (ie changes done by the new thread affect the
 * calling threads APD).
 *
 * Return: The new thread ID or THREAD_NULL on error.
 * errors: ST_CLIST, ST_INFO, ST_PROT, ST_OVFL.
 */

int
ThreadInfo ( mthreadid_t thread, threadinfo_t * info /* INOUT */ );
/*
 * Get information about a child thread, copied into the structure pointed to
 * by info.
 *
 * Return: zero if successful, non-zero otherwise.
 * errors: ST_THR, ST_PROT.
 */

int
ThreadDelete ( mthreadid_t thread, int status, bool adopt );
/*
 * Terminates "thread" with an exit value of "status", THREAD_MYSELF
 * terminates the caller.
 * adopt specifies whether children of this thread will still live.
 *
 * Return: zero if successful, otherwise !=0
 * errors: ST_PROT, ST_THR.
 */

int
ThreadSleep ( mthreadid_t thread, time_t time );
/*
 * Make the thread sleep for the amount specified in time in
 * nanoseconds. A value of zero specifies a thread yield. A value of
 * SLEEP_INFINITY suspends the thread until a ThreadResume() system call is
 * performed on the thread.
 *
 * Return: zero if successful, otherwise !=0
 * errors: ST_PROT, ST_THR.
 */

int
ThreadResume ( mthreadid_t thread );
/*
 * Resumes sleeping "thread"
 *
 * Return: zero if successful, otherwise !=0
 * errors: ST_PROT, ST_THR.
 */

mthreadid_t
ThreadWait ( mthreadid_t thread, int * status /* OUT */ );
```

```
/*
 * Waits for "thread" to exit, THREAD_ANY means any thread.
 * Returns through "status" the exit status, as specified in the
 * ThreadDelete system call.
 *
 * Return: ID of the exited thread. THREAD_NULL on error.
 * errors: ST_PROT, ST_THR.
 */

mthreadid_t
ThreadMyID ( void );
/*
 * Returns ID of calling thread
 *
 * Return: Callers thread ID.
 * errors: Always successful.
 */



/**************\
 * Exceptions *
\**************/

excpthndlr_t
ExcptReg ( excpt_t exception, excpthndlr_t handler );
/*
 * Registers "handler" for "exception".
 *
 * Return: previous handler or NULL if none was registered.
 * errors: ST_EXCPT
 */

int
GetLastError(void);
/*
 * Returns error status from last system call for the thread.
 * errors: Always successful.
 */


/*****************\
 * Page Mappings *
\*****************/


int
PageCopy ( const void * from, void * to, uint n_pages );
/*
 * Copy memory starting at address "from" to the destination "to", both of
 * which must be page aligned. A total of "n_pages" pages are
 * copied. Copy-on-write is used where possible.  The "from" and "to" ranges
 * must be disjoint. Each range must be fully contained in a single object.
```

```
 *
 * Return: zero if successful, otherwise !=0
 * errors: ST_PROT, ST_RNG, ST_NULL.
 */

int
PageMap ( const void * from, void * to, uint n_pages,
          access_t mode, bool fault_in );
/*
 * Make "n_pages" of virtual address space starting at "to" an alias for the
 * virtual address space starting at "from". The address range starting at "to"
 * will be accessible according to "mode". If "fault_in" is true, non-resident
 * pages in the "from" range are faulted in, otherwise the operation is a no-op
 * on such pages.  The two address ranges must either be disjoint or
 * identical. Each range must be fully contained in a single object.
 *
 * Return: zero if successful, otherwise !=0
 * errors: ST_PROT, ST_RNG, ST_NULL.
 */

int
PageUnMap ( void * page, uint n_pages, pagedisp_t disp );
/*
 * Invalidate virtual memory mappings for "n_pages" pages starting at
 * "page". If the pages belong to an object handled by a user-level pager,
 * their mappings simply vanish, otherwise "disp" determines the semantics of
 * the operation. If disp is P_DSP_ZERO, the pages are zeroed (by
 * disassociating them from any backing store). If disp is P_DSP_REPLACE, the
 * physical frames are freed (without first flushing dirty pages) but the
 * association with backing store is kept. If disp is P_DSP_KEEP, the mappings
 * are retained but marked as invalid, forcing a page fault on the next access.
 * All pages of the range must belong to the same object. If disp is
 * P_DSP_UNALIAS all aliases from the specificied page range (which must be
 * part of a single object) are removed. This is a no-op on unaliased pages.
 *
 * Return: zero if successful, otherwise !=0
 * errors: ST_PROT, ST_RNG, ST_NULL.
 */

int
PageFlush ( const void * page, uint n_pages );
/*
 * Ensure that the designated range of pages is clean, i.e. all changes are
 * flushed to disk. Also ensures that association with backing store of pages
 * handled by the default pager is recorded on stable storage. The whole page
 * range must belong to a single object.
 *
 * Return: zero if successful, otherwise !=0
 * errors: ST_PROT, ST_RNG, ST_NULL.
 */
```

```
/**************\
 * Semaphores *
\**************/

int
SemCreate ( void * address, int value, int flags );
/*
 * Creates a semaphore identified by "address" and initialised to "value".
 * RW access is requried at address.
 *
 * Return: zero if successful, otherwise !=0
 * errors: ST_USE, ST_PROT, ST_SEMLMT.
 */

int
SemDelete ( void * address );
/*
 * Destroys the semaphore identified by "address".
 *
 * Return: zero if successful, otherwise !=0
 * errors: ST_PROT, ST_SEMA.
 */

int
SemWait ( void * address );
/*
 * Performs a wait operation on the semaphore identified by "address".
 *
 * Return: zero if successful, otherwise !=0
 * errors: ST_PROT, ST_SEMA, ST_SDEL.
 */

int
SemSignal ( void * address );
/*
 * Performs a signal operation on the semaphore identified by "address".
 *
 * Return: zero if successful, otherwise !=0
 * errors: ST_PROT, ST_SEMA.
 */


void
SeedyPrint ( char *msg );


#endif /* __MUNGI_SYSCALLS_H */
```

Appendix B

Changes to Previous API Versions and Open Issues

B.1 Open issues

The following issues are open/undefined at the moment, or scheduled for revision in the near future:

- Major issues:
  - There is presently no model defined for distribution.
  - The model for persistence is undefined.
  - Startup/restart conventions are undefined.
  - There is presently no device driver model defined. Drivers in the present implementation are hacks.
  - No model for interrupt handling is defined at present.
  - The model for page fault handlers and mappings (Chapter 5) is unproven and incomplete. Specific shortcomings:
    - it doesn't specify how to turn off write protection,
    - it doesn't specify how to control execute permission,
    - it doesn't specify how it is to be used for pinning pages.
  - The thread model is overly simplistic and needs revision, in particular with respect to resource management and priorities.
  - A model for managing/prioritising RAM is missing.
  - Mandatory access control is missing.
- Minor issues:
  - The API lacks a mechanism for enquiring general system info ("kernel information object").
  - There is presently no mechanism defined for supporting profiling.
  - The *GetLastError* syscall sux. It is out of character, doing almost nothing. There should be a better way of achieving the same.
  - There should be access control over *ObjInfo*, but requiring any of read, write or execute rights is too strong.

  - The mechanism for manipulating passwords (*ObjInfo* for reading *and ObjPasswd* for writing) is uncool.
  - The access rights are not orthogonal, leading so some irregularities.

B.2 API changes from version 1.1 to 1.2

- Main change is the merging of APDs and tasks:
  - Introduced protection domain objects, which are templates for APDs.
  - Introduced special objects as a generalisation of bank accounts and protection domain objects.
  - Removed the concept of a *task*.
  - *ThreadCreate* can optionally create a thread in a different protection domain.
  - Consequently replaced task hierarchy by a thread hierarchy.
  - Threads have default bank accounts.
  - Capability handers have been removed.
- Minor changes:
  - Removed the *may move* argument from the *ObjResize* system call.
  - Introduced the *GetLastError* system call.
  - Changed the semantics of the semaphore operations to minimise overheads, particularly for threads waiting on many semaphores.
  - Some rationalisation of types.
  - Slight change in calling convention of *ObjCreate*, *ObjCrePdx*, *ObjInfo*, *ApdGet*, *PdxCall*, *ThreadSleep*.
  - Merged *ObjCrePasswd* and *ObjDelPasswd* into *ObjPasswd.*
  - Changed the semantics of slot locking, with consequent effects on *ApdInsert*, *ApdDelete* and *PdxCall.*
  - The *sharable* flag has been removed from the API spec for the time being, as its use and semantics are presently under consideration.
  - Removed the *ExcptRet* library call.
  - Removed the *PageSize* library call.

B.3 API changes from version 1.0 to 1.1

Note: API version 1.1 never left the draft stage.

- Slight reorganisation and extension of *ObjInfo*.
- Negative capabilities introduced.
- Changed capability parameters to addresses in all system calls. This means that capability are *only* implicitly prevented, the possibility for explicit presentation no longer exists. This helps confinement, as it is easily possible to construct protection domains which do not grant access to the clists defining them, and thus prevent their modification.

- Introduced locking of individual APD slots.
- Each APD slot can now hold a Clist **and** a handler reference.
- Interchanged meaning of zero and negative values for *n_pd* parameters *PdxCall* and *TaskCreate*.
- Reduced number of arguments passed to new thread in *PdxCall* and *TaskCreate* to one (capability-sized). Dropped the *stack_size* parameter.

Appendix C

GNU Free Documentation License

Version 1.1, March 2000

Preamble

The purpose of this License is to make a manual, textbook, or other written document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

C.1 Applicability and Definitions

This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you".

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (For example,

if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup has been designed to thwart or discourage subsequent modification by readers is not Transparent. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML designed for human modification. Opaque formats include PostScript, PDF, proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

C.2 Verbatim Copying

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

C.3 Copying in Quantity

If you publish printed copies of the Document numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a publicly-accessible computer-network location containing a complete Transparent copy of the Document, free of added material, which the general network-using public has access to download anonymously at no charge using public-standard network protocols. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

C.4 Modifications

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has less than five).
- State on the Title page the name of the publisher of the Modified Version, as the publisher.
- Preserve all the copyright notices of the Document.
- Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
- Include an unaltered copy of this License.
- Preserve the section entitled "History", and its title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
- Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a

work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

- In any section entitled “Acknowledgements” or “Dedications”, preserve the section’s title, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

- Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

- Delete any section entitled “Endorsements”. Such a section may not be included in the Modified Version.

- Do not retitle any existing section as “Endorsements” or to conflict in title with any Invariant Section.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version’s license notice. These titles must be distinct from any other section titles.

You may add a section entitled “Endorsements”, provided it contains nothing but endorsements of your Modified Version by various parties – for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

C.5 Combining Documents

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections entitled “History” in the various original documents, forming one section entitled “History”; likewise combine any sections entitled “Acknowledgements”, and any sections entitled “Dedications”. You must delete all sections entitled “Endorsements.”

C.6 Collections of Documents

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

C.7 Aggregation With Independent Works

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, does not as a whole count as a Modified Version of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an “aggregate”, and this License does not apply to the other self-contained works thus compiled with the Document, on account of their being thus compiled, if they are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one quarter of the entire aggregate, the Document’s Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

C.8 Translation

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

C.9 Termination

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

C.10 Future Revisions of This License

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> Copyright © YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have no Invariant Sections, write "with no Invariant Sections" instead of saying which ones are invariant. If you have no Front-Cover Texts, write "no Front-Cover Texts" instead of "Front-Cover Texts being LIST"; likewise for Back-Cover Texts.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.