The Walnut Kernel: User Level Programmer's Guide

Maurice Castro[1]

TECHNICAL REPORT NO. 95/222

May 1995

revised
November 1995

[1]Computer Science, Monash University, Clayton, Victoria 3186, AUSTRALIA
maurice@cs.monash.edu.au

Abstract

The **Walnut Kernel** is a capability-based operating system under development in the Department of Computer Science at Monash University.

The **Walnut Kernel** employs 128-bit names - Password-Capabilities - for views onto persistent objects. The random allocation of names within a sparse name space provides a known level of statistical security for views and the contents of objects. Associated with each name is a set of rights which entitle the holder of the capability to access a section of the named object in a specified way.

This document contains a programmer's manual for the **Walnut Kernel**. It provides a brief outline of the basic concepts of the kernel and detailed descriptions of the process environment and system calls provided by the kernel.

Contents

1 Overview **3**

2 Objects **4**

3 Capabilities **5**
3.1 View 5
3.2 User Rights 5
3.3 System Rights 5
3.4 Deriving Capabilities 6

4 Process Structure **7**
4.1 Process Address Space 7
4.2 Parameter Page 9
4.2.1 Parameter Block 9
4.2.2 Message Block 12
4.3 The Wall 14

5 Process Structure Conventions **15**
5.1 The Process Object 15
5.2 The Process 15

6 Process Creation **17**
6.1 Making Processes 17
6.2 Initial Process State 17

7 Subprocess Zero **20**
7.1 Freeze 20
7.2 Thaw 21
7.3 Wakeup 21
7.4 Cooee 21
7.5 Protected Freeze 21
7.6 Protected Thaw 22

8 Subprocesses **23**
8.1 Anatomy of a Subprocess 23
8.2 Operations on Subprocesses 23
8.3 Scheduling 23

9 Messages and Mailboxes **25**
9.1 Sending Messages 25
9.2 Receiving Messages 25
9.3 Mailboxes 25

10 Exceptions **27**
10.1 Types of Exception 27
10.2 Trap Handling Subprocesses 27
10.3 The Trap Message 28

11 System calls **30**
11.1 Procedure 30
11.2 Available System Calls 31
11.2.1 Make Object 31
11.2.2 Derive Capability 32
11.2.3 Delete Capability 33
11.2.4 Delete Derived Capabilities 33
11.2.5 Resize Object 34
11.2.6 Shrink Object 34
11.2.7 Wait 35
11.2.8 Load Capability 36
11.2.9 Unload Capability 37
11.2.10 Identify Capability 38
11.2.11 Make Process 39
11.2.12 Send Message 42
11.2.13 Receive Message 43
11.2.14 External Send Message 43
11.2.15 External Read Memory 44
11.2.16 External Write Memory 44
11.2.17 Bank 45
11.2.18 Restrict Rights 46
11.2.19 Capability Status 47
11.2.20 Rename Capability 48
11.2.21 Make Subprocess 49
11.2.22 Delete Subprocess 50
11.2.23 Load Register Set 50
11.2.24 Save Register Set 50
11.2.25 Set Trap 51
11.2.26 Receive Message and Close Box 51
11.2.27 Accept Mail 52
11.2.28 Close Mail Box 52
11.2.29 Copy Object 53
11.2.30 Check Process State 54
11.2.31 Set Heir of Process 54

1 Overview

The **Walnut Kernel** is a capability-based operating system under development in the Department of Computer Science at Monash University. This operating system draws on the concepts of and experience gained from the Password-Capability System[2].

The **Walnut Kernel** employs 128-bit names - Password-Capabilities - for views onto persistent objects. The random allocation of names within a sparse name space provides a known level of statistical security for views and the contents of objects. Associated with each name is a set of rights which entitle the holder of the capability to access a section of the named object in a specified way.

The **Walnut Kernel** was designed as a portable operating system although it currently runs only on 80486 based PCs. Programs are compiled on a FreeBSD 1.1 system and transferred onto the target machine on floppy disks. Work is continuing on the development of the kernel as well as the development of interfaces, shells, and utilities for the system.

This document contains a programmer's manual for the **Walnut Kernel**. The document is subject to revision as the kernel alters and currently describes only the lowest level of the kernel interface.

Acknowledgments

The author would like to acknowledge the following contributions:

- Prof C.S. Wallace - co-author of the kernel
- Mr Glen Pringle - author of many of systems utility programs
- Mr Carlo Kopp - author of the UNIX compatibility libraries

The 'Secure RISC Architecture' project is supported by a grant from the Australian Research Council (A49030623). Maurice Castro is a recipient of an Australian Postgraduate Research Award.

[2]M. Anderson, R D. Pose, and C S. Wallace. A Password-Capability system. *The Computer Journal*, 29(1):1–8, 1 1986.

2 Objects

All entities controlled by the **Walnut Kernel** are objects. A **Walnut Kernel** object is analogous to a segment in segmented computer architecture. It comprises an ordered array of bytes. An individual byte is identified by its 'offset', a number indexing the array. The first byte has offset zero. An object is defined by the following characteristics:

Maximum Offset The largest addressed offset in the object. (Note: this value is set on creation and automatically increases, as long as there are pages available in the allocated space of the object)

Limit The largest addressable offset allowed in the object.

Maximum Size The maximum number of bytes guaranteed to be available to an object. The number of bytes includes storage for the objects capabilities and dope vectors. (In practice this value represents the maximum number of pages and header pages guaranteed to be available to an object. The number of pages is calculated by dividing the maximum size by the page size and rounding upwards.)

Maximum Capabilities The maximum number of capabilities that can represent this object. (Note: this value automatically increases, as long as there is space available to hold the new derived capabilities)

Money The amount of money the object has available. Sufficient money must be present in an object to pay for its resource consumption.

Each object has at least one capability that allows access to the object - the object's **Master Capability**. Deletion of the object's master capability results in the deletion of the object. Other capabilities for views of the object are derived from the master capability or its descendants.

3 Capabilities

The **Walnut Kernel** employs password capabilities to identify access rights to objects. Each capability (see figure 1) consists of a 128 bit identifier composed of four 32 bit values: a volume number, a serial number, password 1 and password 2. Associated with each capability is a view which determines the region of an object a capability applies to, a set of user rights, and a set of system rights which control how that capability is to be used.

32 *bits*	32 *bits*	32 *bits*	32 *bits*
Volume	Serial	Password 1	Password 2

Figure 1: A Password Capability

3.1 View

A view is the attribute of a capability that defines the region of the object that can be addressed by the possessor of the capability. Views are contiguous regions and are defined by an offset from the base of the object and an extent. The view entitles the user to address part of an object, it does not guarantee that pages are contained in that region nor that the pages are readable by user processes.

3.2 User Rights

User rights consist of a set of 32 bits which are managed by the kernel. The kernel attaches no meaning to the user rights bits. They are intended to be used by user processes to implement access to services in a way that is analogous to the control system rights bits have over access to kernel services.

3.3 System Rights

The system rights associated with a capability are encoded in a 32-bit word, basically as the OR of bits representing particular rights (the SRSEND field is an exception). For the numeric value of the system rights symbols see figure 4.

SRDERIVE - Allow capabilities to be derived from this capability.

SRSUICIDE - Allow this capability to destroy itself and its children.

SRDEPOSIT - Allow the holder of this capability to deposit money into the object.

SRWITHDRAW - Allow the holder of this capability to withdraw money from the object.

SRREAD - Allow the holder of this capability to read from the view.

SRWRITE - Allow the holder of this capability to write to the view.

SREXECUTE - Not used.

SRUSER - Allow user processes to use the view.

SRPEEK - Allow the holder of this capability to perform a peek system call on the process represented by this capability (see 11.2.30).

SRMULTILOAD - Allow this capability to be loaded by any process. If this right is absent then only processes with a serial number equivalent to the capability's password 2 may load this capability.

SRSEND - an 8-bit field which, if non-zero, specifies the subprocess to which messages may be sent by using this capability. This field has two special values: `0xff` - allow messages to be sent to any subprocess of the process, and `0xfe` - disallow messages to subprocess zero but allow messages to be sent to any other subprocess of the process.

3.4 Deriving Capabilities

Derived capabilities have equal or lesser rights than than their parent capability, at the time of derivation. *Suicide right* is an exception as this right may be added to the children of capabilities which do not hold this right.

The rights of a parent capability may be reduced through the use of the *restrict* system call after a child capability has been derived. The child capability is unaffected by the restriction of the parent capabilities rights.

4 Process Structure

A process in the **Walnut Kernel** is essentially an object which contains state information relating to the execution of the process. The minimal information found within a process object is:

- Sub-process table
- Message slots
- Table of Loaded Capabilities
- Process cash
- Lock words
- Parameter page
- Address map

Of these only the parameter page and the address map are directly accessible to the user process. The address map is read-only. The parameter block and the remainder of process object are both readable and writable by the user process.

The process structure is detailed in diagram figure 2.

4.1 Process Address Space

The address space of a process operating under the **Walnut Kernel** is composed of three regions:

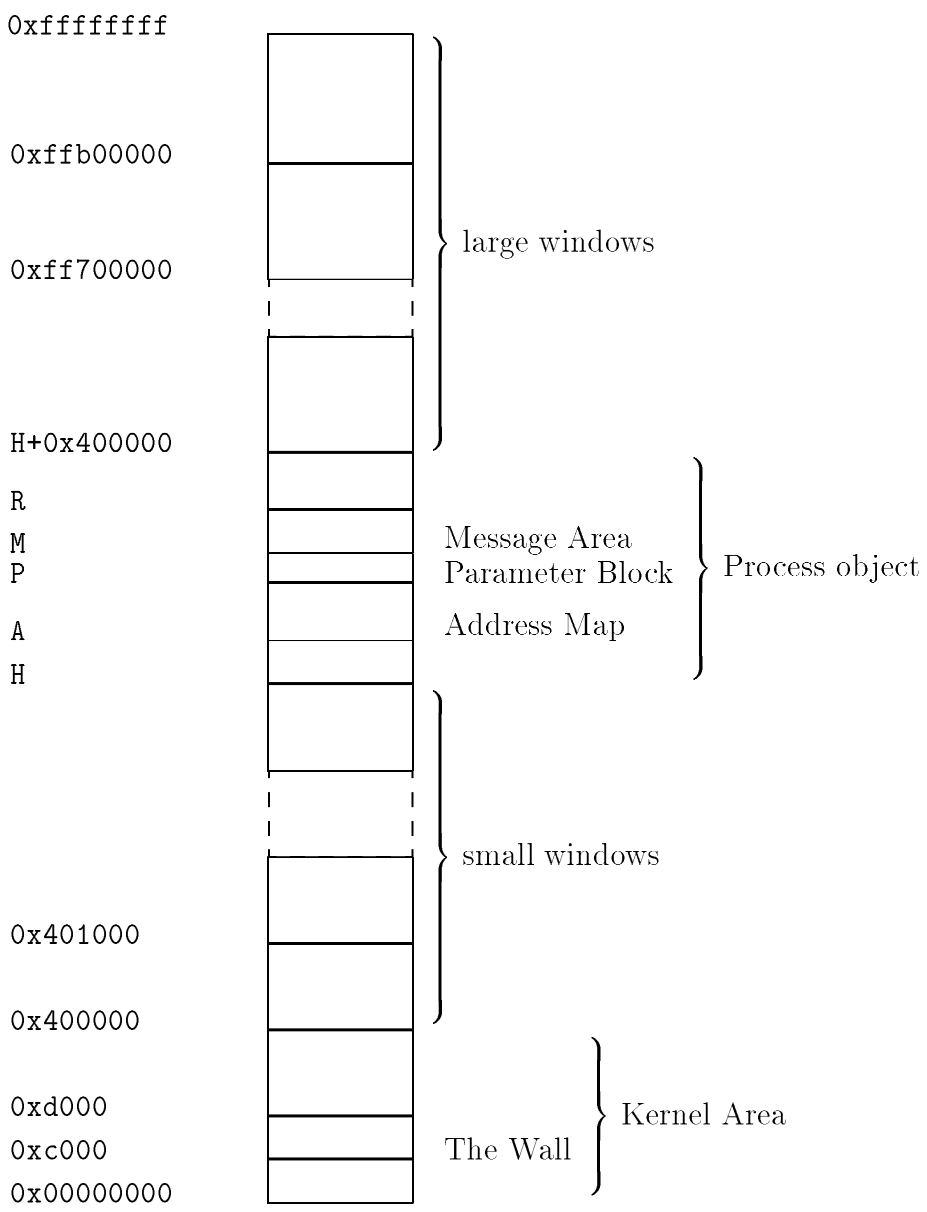
Kernel Area is located at the bottom of the address space and is not addressable by user processes.

Small Window Area is located above the Kernel Area and has a page sized granularity. Single pages or multiple pages of objects may be mapped into this region of the address space by a user process. These mapped regions always begin and end on a page boundary.

Large Window Area is located above the Small Window Area. It has a coarser granularity than the Small Window Area. The first large window contains the Process Object. All other large windows are allocated by the process.

On a system with a 4 kilobyte page size, large windows have a granularity of 4 megabytes and small windows have a granularity of 4 kilobytes.

Two distinct paradigms are used to describe how the address is populated with objects.



	Description	Constant Name	Value
R	remainder of process object	PARAMADDRESS + `0x1000`	`0x1011000`
M	message area	EXTRAADDRESS	`0x101004c`
P	parameter page	PARAMADDRESS	`0x1010000`
A	address map		`0x100f000`
H	process header	PROCHDADDRESS	`0x1000000`

Figure 2: **Process Address Space**: This diagram describes the major features of the address space seen by a process operating on a system with 4 kilobyte pages. The message area and the parameter block are collectively known as the parameter page.

The Password-Capability system used the term ‘Window Registers’ to describe a set of segment registers. The Password-Capability system used the upper bits of the virtual address to indicate which register was in use. We retain this terminology in the **Walnut Kernel**. The analogy between the two systems is imperfect as: the **Walnut Kernel** supports two classes of window registers; and although the number of window registers is fixed on the **Walnut Kernel** the location of the registers in the virtual address space is not fixed, under the Password-Capability System both of these parameters were constant.

The second paradigm describes the operation of the system. On a system with 4 kilobyte pages, it views the address space as two address ranges. The address range from `0x400000` to `0xffffff` can have objects loaded on 4 kilobyte boundaries. The address range `0x1400000` to `0xffffffff` has objects loaded on 4 megabyte boundaries.

4.2 Parameter Page

This page is composed of two parts: the parameter block and the message area. The parameter block is a structure defined in the file **param.h**. This block is used to pass parameters to the kernel when a system call is made. The second area is used to pass additional information to the kernel and to receive information from the kernel. The information passed via the message area varies with the type of call.

4.2.1 Parameter Block

The declaration of the parameter block structure is found in figure 3.

The fields are named after the function they are used for in the majority of calls. The values contained in the fields are:

error The error field contains an integer error value on returning from a system call. If the value is zero then the system call completed successfully. If the value is greater than zero the system call could not be completed successfully. If the return value is negative[3] or greater than 20000000_{10} an internal kernel error has occurred: contact the system’s maintainer urgently and report the value. The file *include/kerror.h* contains a translation table which allows error values to be converted to ascii strings.

vol serial pass1 pass2 The capability field composed of vol, serial, pass1 and pass2, contains either a capability being passed to a system call or a capability being passed back by a system call.

srights The system rights field contains one of

[3]Negative error values are used internally by the kernel to indicate partial completion of a system call which cannot be completed because of a transient problem.

```
/* -------------------------------------------------------------- */
/*                     Parameter Structure                        */
/* -------------------------------------------------------------- */
typedef struct Paramst {
    Sw error;
    Uw vol;          /* volume ID */
    Uw serial;       /* serial in volume */
    Uw pass1;        /* password 1 */
    Uw pass2;        /* password 2 */
    Uw srights;      /* System rights */
    Uw urights;      /* User rights */
    Sw base;         /* Offset of cap from front of object */
    Sw limit;        /* Max addressing offset from base */
                     /* Zero means "to end of object" */
    Sw money;        /* money word */
    Uw type;         /* Type of object */
    Sw maxoff;       /* Max addressing offset in whole object */
    Sw maxsz;        /* Max size of defined content */
    Sw maxcap;       /* Max capabilities now allowed */
    Sw offset;       /* An offset in a capability window */
    Sw subpn;        /* A subprocess number */
    Sw cindex;       /* Index of a capl in a process TLC */
    Uw clocktime;
    Uw reserve;      /* Non-zero shows reserved by sub-process */
    } Param;
```

Figure 3: **Parameter Block Declaration**

- a bitmap indicating the system rights provided by a capability (Figure 4 defines the symbolic and numeric forms of the system rights bits)
- a mask which restricts the system rights provided to a derived capability
- a set of limits used in the creation of a process
- an encoded process state when inquiring about a process state

urights The user rights field contains either

- a bitmap indicating the user rights provided by a capability
- a mask which restricts the user rights provided to a derived capability

base The base field contains an offset from the beginning of a view or - on process creation - the time, in seconds, at which the new process is scheduled to wake up.

limit The limit field contains one of

- the length of a message
- the maximum addressable offset of an object
- the maximum size of a view

money The money field contains one of

- the amount of money in an object
- the amount of money to be deposited or withdrawn
- the amount of money to be sent with a message
- the amount of money received from a message

type The type field contains the object type. The top bit of this field is set if the object is a process. The following type values are reserved by the kernel:

`00000000`	
`00000003`	Prototype process
`0000ffff`	Physical memory object
`80000000`	
`80000002`	Drive process
`80000003`	Prototype process

maxoff The maximum offset field contains the current maximum offset of an object.

maxsz The maximum size field contains the current maximum size of an object.

maxcap The maximum capability field contains the current maximum number of capabilities for a process

offset The offset field contains an offset into a process's address space.

subpn The subprocess number field contains the destination subprocess number for a message.

cindex The capability index field contains the index into the table of loaded capabilities that a capability occupies.

clocktime The clocktime field, after returning from a system call, contains the current time in seconds. The clocktime field is set to the wakeup time for a process when a wait system call is made.

reserve The reserve field provides both a locking function which prevents other subprocesses accessing the parameter block and indicates the type of kernel call being made. The currently available kernel call constants are listed in figure 5.

```
#define SRDERIVE    0x40000000  /*  System rights bits   */
#define SRSUICIDE   0x20000000
#define SRDEPOSIT   0x10000000
#define SRWITHDRAW  0x08000000
#define SRREAD      0x04000000
#define SRWRITE     0x02000000
#define SREXECUTE   0x01000000
#define SRUSER      0x00800000
#define SRPEEK      0x00400000
#define SRMULTILOAD 0x00200000
#define SRSEND      0x000000FF  /* Bits relating to send rights */
```

Figure 4: **System Rights Constants**

4.2.2 Message Block

The message area's contents are interpreted differently for each class of call. There are currently three classes of information stored in the message block:

- Messages - Messages to be sent by the send message system call and messages recovered by the receive message system call are stored at the front of the message block.

```
/* -------------------------------------------------------------- */
/*                          Action Codes                           */
/* -------------------------------------------------------------- */

#define K_MAKEOBJ 1
#define K_MAKECAP 2
#define K_DEL 3
#define K_DELDER 4
#define K_RESIZE 5
#define K_SHRINK 6
#define K_WAIT 7
#define K_LOADCAP 8
#define K_UNLOADCAP 9
#define K_CAPID 10
#define K_MAKEPROC 11
#define K_SEND 12
#define K_RECV 13
#define K_EXTSEND 14
#define K_EXTREAD 15
#define K_EXTWRITE 16
#define K_BANK 17
#define K_RESTRICT 18
#define K_CAPSTAT 19
#define K_RENAME 20
#define K_MAKESUBP 21
#define K_DELSUBP 22
#define K_LOADREG 23
#define K_SAVEREG 24
#define K_SETTRAP 25
#define K_RECV_CLOSE 26
#define K_ACCEPT_MAIL 27
#define K_CLOSE_BOX 28
#define K_COPYOBJ 29
#define K_PEEK_PROC 30
#define K_SET_HEIR 31
```

Figure 5: **Defined Kernel Call Constants**

- System states - The save register and load register system calls store the register set and other state information for a subprocess at the front of the message block.
- Read/Write Data - Bytes to be transferred by the external read or external write system calls are stored at the front of the message block.
- Initialization information - Initial values used to set stack and program counters, the name of an heir and a list of capabilities to be pre-loaded into a process are stored at the front of the message block.

4.3 The Wall

Every process has a read-only page mapped into its address space known as **the Wall**. This page contains public information, including the current time, and the capabilities of public utilities. A **wall manager** places information in the wall. The wall currently contains:

`0xc000`	Scheduler Start Variable†
`0xc004`	Physical Object: Volume†
`0xc008`	Physical Object: Serial†
`0xc00c`	Physical Object: Password 1†
`0xc010`	Physical Object: Password 2†
`0xc000`	GLui: Magic Number
`0xc004`	GLui: Volume
`0xc008`	GLui: Serial
`0xc00c`	GLui: Password 1
`0xc010`	GLui: Password 2
`0xc014`	Name Server Set: Magic Number
`0xc018`	Name Server Set: Volume
`0xc01c`	Name Server Set: Serial
`0xc020`	Name Server Set: Password 1
`0xc024`	Name Server Set: Password 2
`0xc028`	Name Server Set: Offset
`0xcfe0`	Time in Seconds
`0xcfe4`	Time in Microseconds

†: These locations are used by the initialization process. After initialization the capability of the physical object is overwritten by the initialization process and the value of the scheduler start variable is no longer significant.

5 Process Structure Conventions

This section covers the conventional layout of a process (Section 4 outlined the mandatory elements of a process structure).

5.1 The Process Object

The following elements of the process object are visable to the user process and are provided by the kernel:

- The Process Address Map
- The Parameter Page
- The Message Area

They form part of the mandatory component of the process structuring convention used by **Walnut Kernel** processes.

By convention the following items are located within the process object

Startup Code Area (optional) This area may contain a small amount of code used in starting a process.

File Descriptor Table (mandatory) This area contains the file descriptors for use by the process. Note: The first 3 elements of the File Descriptor Table are mandatory to allow for standard output, standard input and standard error.

Private Data Pointer Table (mandatory) This area contains pointers to private data. The table is indexed by the capability index of the executing code and is used to locate data used by the executing code.

Default Heap (optional) The default location for the creation of the heap.

Default Stack (optional) The default location for the creation of the stack.

The structure of the process object is outlined in figure 6.

5.2 The Process

Conventional processes will be constructed according to the following rules:

- The code object will be loaded at address `0x1400000`
- The data object will be loaded at address `0x5400000`
- Initialized data will be placed at the front of the data object

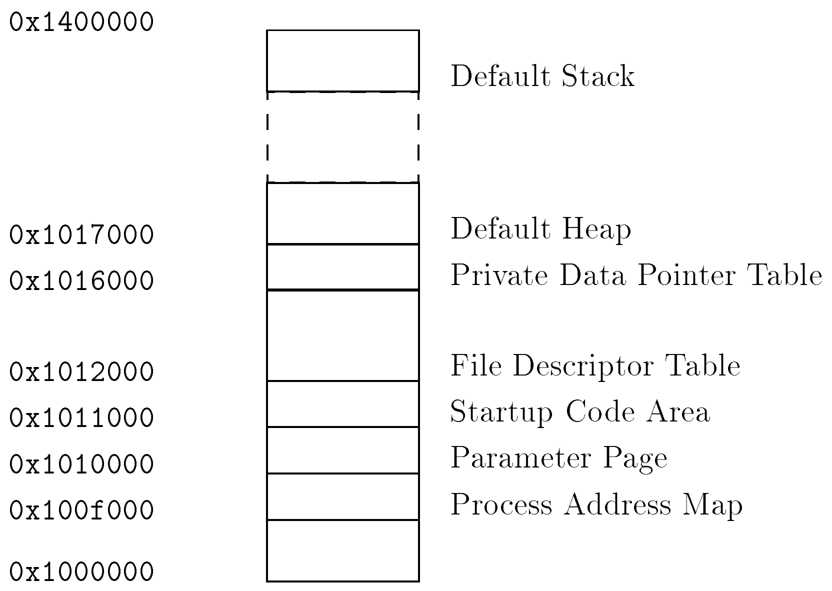


Figure 6: **Process Object**: This diagram describes the major features of the process object

This design allows multiple instances of a process to be created by sharing the code objects and using copies of the data objects. In addition by placing the initialized data at the front of the data object it is possible to ensure that the original data object is compact and hence easy to copy. The copy of the data object will expand as required when uninitialized data is accessed.

This arrangement of code and data allows up to 64 Mbytes of code to be supported. With the introduction of shared code libraries larger programs can be supported.

6 Process Creation

This section describes the process of creating a process and the initial state of a new process.

6.1 Making Processes

A process is created using the **Make Process** call covered in section 11.2.11. This section will provide a general introduction to the creation of a process.

Creating a process involves:

- Creating a new process object
- Creating an address space
- Loading the new process object into the address space
- Creating subprocess 0 and subprocess 1.
- Loading pre-loaded capabilities into the address space
- Setting initial program counter and stack pointer values for subprocess 1.
- Setting the wake up time of subprocess 1.
- Loading the new process object into address space of the creating process

This process appears as an atomic operation to the process issuing the **Make Process** system call. If the system call was successful the master capability for the new process object will be returned and the new process will be loaded at the address given in **offset** in the parameter block.

At the completion of the **Make Process** system call, the new process object is loaded into the address space of the creating process. If the new process object is larger than 4 megabytes in size, only the first 4 megabytes of the new process object is visable. Thus the process which issued the **Make Process** system call and the new process have a region of shared memory.

6.2 Initial Process State

Immediately after a process has been created:

- The parameter block of the new process will contain:

vol	Volume of master capability for process
serial	Serial of master capability for process
pass1	Password 1 of master capability for process
pass2	Password 2 of master capability for process
srights	Encoded process creation parameters
urights	User rights of process's master capability
limit	Maximum size of process object (hard limit)
money	Amount of money in process object / process cash
type	Type of process
maxoff	Maximum offset of view on process object
maxsz	Maximum size of process object
maxcap	Maximum number of capabilities

- The message area will consist of a table of pre-loaded capabilities with the format:

vol	Volume
serial	Serial
pass1	Password 1
pass2	Password 2
base	Start of the loaded window relative to the capability
limit	Size of the loaded window. Zero indicates capability limit
offset	Location of window in the new process's address space
cindex	Index in table of loaded capabilities. Zero for automatic allocation

- The process object will be loaded at the location PROCHDADDRESS
- The address space will contain all pre-loaded capabilities
- Only subprocess 0 and subprocess 1 will exist
- Subprocess 1 will begin executing

The process creation parameters are encoded in the system rights field:

8 bits	*8 bits*	*8 bits*	*8 bits*
Max subp	# message slots	Max loaded caps	# auto load caps

msb lsb

- Max subp - The maximum number of subprocesses for the new process including subprocess 0.
- # message slots - The number of message slots for the new process. As a message slot is reserved for subprocess 0 the number of message slots must be 1 or greater.

- Max loaded caps - The maximum number of loaded capabilities for the new process. This number includes the capability for the process.
- # auto load caps - The number of capabilities to be automatically loaded into the new process's address space including the capability for the new process.

When a process is created two equal sums of money are deposited into the new process. The sums are deposited into the process cash and the process object respectively. The size of one of the deposited sums is reported in the money field.

It is normal practice for the first action of a process to be the duplication of the information passed in at process creation. It is particularly important to store the capability for the process as it is not possible to locate the master capability for the process subsequently.

The creation of subprocesses other than subprocess 0 and subprocess 1 is handled by the application using the Make Subprocess call.

7 Subprocess Zero

The **Walnut Kernel** implements two direct methods of communication with the kernel: system calls and messages to subprocess zero of a process. The system call mechanism (described in the section 11) allows a process to alter its own state, operate on capabilities and send messages. The subprocess zero mechanism allows a process to control another process's state.

Subprocess zero functions are accessed by sending messages to a process's subprocess zero. The message contains a function identifier and arguments. On receipt of a message to subprocess zero the kernel interprets the instruction provided and performs the required action. Subprocess zero operations and messages are the highest priority function of a process.

The currently implemented subprocess zero functions are:

Freeze Prevent process from being scheduled.

Thaw Allow process to be scheduled.

Wakeup Set the wakeup time of the specified subprocess to zero

Cooee Request the process to send a status message using a specified capability.

Protected Freeze Prevent process from being scheduled until all protected freezes on the process have been thawed.

Protected Thaw Allow a process to be scheduled when all other protected freezes have been thawed.

Figure 7 lists the identifiers and arguments of the messages.

Function	Function ID	Arg 1	Arg 2	Arg 3	Arg 4
Freeze	33330001				
Thaw	33330002				
Wakeup	33330003	subp #			
Cooee	33330004	vol	ser	pass 1	pass 2
Prot Freeze	33330007	magic			
Prot Thaw	33330008	magic			

Figure 7: Subprocess Zero Functions and Arguments

7.1 Freeze

On receipt of a freeze message subprocess zero sets the process state to frozen and causes the process to be removed from the scheduler queue.

7.2 Thaw

When a process receives a message it is placed into the scheduler queue. If the process is frozen the process is typically removed from the queue after the subprocess zero messages are parsed. On receipt of a thaw message, subprocess zero sets the process state to normal and process execution resumes.

7.3 Wakeup

The wakeup message sets the wakeup time of the nominated subprocess to the current time. This allows a process to start a process that has suspended activity and has closed mail boxes as the mail box allocated to subprocess zero cannot be closed.

One application of this function is to allow the initialization of data structures within a process object. The process is created with a wakeup time of never preventing the scheduling of the process. The creating process initializes the required data structures before waking the created process up. At that stage the created process may elect to open its mail boxes as processes are created with all but subprocess zero's mail boxes closed.

7.4 Cooee

On receiving a cooee message subprocess zero attempts to send a message using the capability found in the cooee message. If the capability in the cooee message allows transmission to any subprocess of a process then the message will be sent to subprocess one of the nominated process, otherwise, the message will be sent to the subprocess represented by the capability.

The reply message is of the form:

33330005	volume	serial	status

The message consists of a set of words which represent the Cooee reply identifier, the volume and serial number of the current process and a process status. The process status is given in figure 8.

7.5 Protected Freeze

On receipt of a protected freeze message subprocess zero sets the process state to frozen, XORs the magic word with a key held in the process state, increments a count held in the process state and causes the process to be removed from the scheduler queue. This prevents other parties from thawing the process unless they know the set of magic words used in the protected freeze operations applied to the process.

State	State ID	Value
Normal State	PROCSTATENORMAL	1
In Kernel Call	PROCSTATEKERNEL	2
In Read Fault	PROCSTATERFAULT	3
In Write Fault	PROCSTATEWFAULT	4
Process Frozen	PROCSTATEFROZEN	5
Process in Probate	PROCSTATEPROBATE	6
Process Dead	PROCSTATEDEAD	7

Figure 8: Process Status

7.6 Protected Thaw

On receipt of a protected thaw message subprocess zero XORs the magic word with a key held in the process state and decrements a count held in the process state. If both the count and key held in the process state are zero then the process is thawed. If the count is zero and the key is non-zero then the process is terminated.

8 Subprocesses

Subprocesses are implemented in the **Walnut Kernel** as threads of execution which share a single address space. This section describes subprocesses and their scheduling.

8.1 Anatomy of a Subprocess

When a process is created a fixed number of subprocess slots are allocated in the process structure. These slots form the *subprocess table* which is used to store the subprocess states.

When a subprocess is created the creator specifies a priority which is used to determine which subprocess should be scheduled, the starting address of the subprocess and the the address of the subprocess's stack pointer. It is the responsibility of the programmer to ensure that the stacks of subprocesses do not overlap.

Subprocesses share the address space of the process and hence have no protection from the actions of other subprocesses of the process.

8.2 Operations on Subprocesses

Subprocesses can be made through the use of the **K_MAKESUBP** system call and they are destroyed by **K_DELSUBP**. Messages are sent to subprocesses using **K_SEND** and **K_EXTSEND**. There are three types of capabilities which can be used to send messages: capabilities which can send messages to any subprocess of a process, capabilities which can send a message to any subprocess of a process other than subprocess zero, and capabilities which can only send messages to a particular subprocess. The type of capability determines if the subprocess parameter of the send operation is used.

8.3 Scheduling

Subprocesses have the semantics of processes on a time sharing system. That is, when a subprocess of a process is executing no other subprocess of that process can be executing. On the **Walnut Kernel** processes are used to support concurrent execution.

The algorithm for determining which subprocess to run at the beginning of a time slice for a process is as follows:

1. If a subprocess was executing and there is a non-zero value in the *reserve* field of the parameter block resume execution of that subprocess.

2. Execute the subprocess with the highest priority which is not waiting.

3. For subprocesses of equal priority select the first subprocess encountered in the subprocess table.

Before performing the algorithm to determine which subprocess to schedule the mail boxes are scanned. If a new message has arrived for a subprocess the subprocess is made runnable (not waiting).

Subprocesses can ensure that other subprocesses of the current process are excluded from executing by setting the *reserve* field to a non-zero value. It is essential that any subprocess attempting to make a system call sets the *reserve* field to the appropriate value for the system call before accessing other elements of the parameter block. It is also necessary to test or copy all required values from the parameter block before zeroing the *reserve* field after returning from a system call.

9 Messages and Mailboxes

This section describes the processes of sending and receiving messages.

9.1 Sending Messages

Messages are sent using either the **K_SEND** or **K_EXTSEND** system calls. A message consists of the contents of the *message area*. The length of the message is variable (currently up to 16 words may be sent) and it is specified by setting the *limit* field to the number of bytes to be transferred.

Messages are sent to processes represented by a capability. The capability may be derived to allow messages to be sent to only one subprocess or to allow messages to be sent to all subprocesses of the process. If the latter type of capability is used then the *subpn* field contains the destination subprocess number.

A message will only be sent if there is an empty mailbox available to receive the message at the destination process. An error is returned if there are no suitable mailboxes at the destination process.

9.2 Receiving Messages

Messages are retrieved and mailboxes are cleared by issuing a **K_RECV** system call. A match string can be specified for the *receive* system call allowing the user program to control the order in which messages are retrieved from mailboxes.

When there is a message in a mailbox waiting to be received, the wakeup time of the subprocess is set to the current time. This nullifies the effect of any **K_WAIT** system calls.

9.3 Mailboxes

Mailboxes have 3 independent parameters which determine whether or not they will accept a message: state, prefix, and subprocess.

The state of the mailbox:

Open - The mailbox is prepared to accept a message that meets the other criteria

Closed - The mailbox will not accept messages

A message prefix consists of a string of characters:

Non-zero length - Only messages starting with the prefix string are accepted. The length of the prefix string is specified in bytes.

Zero Length - Accept any message meeting the other criteria

Mailboxes may accept messages for specified subprocesses:

Subprocess 0 - 250 - Only messages intended for the specified subprocess are accepted

Subprocess 255 - Accept any message meeting the other criteria

If a message matches the mailbox's criteria and the mailbox is empty then the message is placed in the mailbox. The criteria are used to ensure that mailboxes are available for particular types of messages. The first available mailbox that accepts the message is used.

The **K_RECV_CLOSE**, **K_ACCEPT_MAIL** and **K_CLOSE_BOX** system calls are used to manipulate the parameters of the mailbox.

Both the **K_CLOSE_BOX** and the **K_RECV_CLOSE** system calls close mailboxes. The **K_RECV_CLOSE** receives a message from a mailbox and then closes the mailbox from which the message was extracted. The **K_ACCEPT_MAIL** system call opens a mailbox and specifies the parameters which determine the messages the mailbox will accept.

10 Exceptions

This section describes the handling of exceptions by processes under the **Walnut Kernel**. The default behavior of the **Walnut Kernel** is to terminate any process which encounters an exception. This behavior can be modified by using *trap handling* subprocesses.

10.1 Types of Exception

The **Walnut Kernel** detects the following exceptions:

FPFAULT - All exceptions relating to errors in arithmetic. This may include floating point exceptions, integer arithmetic exceptions, dividing by zero, overflow and underflow. The types of errors detected by this exception are processor dependent.

OPFAULT - This exception is raised when an invalid instruction is parsed by the processor.

ADDRSFAULT - This exception is used to catch all errors relating to addresses. It is raised under the following conditions:

- An unmapped region of the address space has been accessed
- A write has been attempted on a read-only area of memory
- A read or write has been attempted on an privileged area of memory
- An object could not be automatically expanded to accommodate the attempted access due to the lack of unreserved space on the volume

DBFAULT - This exception is raised whenever a debug exception is raised by the processor. This exception is processor dependent.

ALIGNFAULT - This exception is raised on unaligned accesses. This exception is processor dependent.

10.2 Trap Handling Subprocesses

A trap handler is a normal subprocess which has been nominated to receive trap messages for a given subprocess. The **K_SETTRAP** system call is used to inform the kernel where trap messages should be sent. The *set trap* system call takes two arguments: the subprocess for which traps are to be handled and the subprocess which will handle the trap.

A subprocess cannot handle its own traps. If a subprocess traps and the trap message is to be sent to the same subprocess then the process will be terminated.

When an exception occurs in a subprocess, which has a nominated trap handler, the subprocess with the fault is marked **DEAD**, its wake up time is set to **NEVER** and a message is sent to the trap handler. The format of the message is discussed in section 10.3.

The trap handler can examine and alter the state of the dead subprocesses register sets through the use of the **K_LOADREG** and **K_SAVEREG** system calls. The subprocess can be restored to operation through the use of the **K_MAKESUBP** system call.

10.3 The Trap Message

A five word message is sent (see figure 9) to the trap handling subprocess. The words of the message are:

1. Message Type - this word indicates that the message is the result of an exception. The *failure message identifier* is `0x3333ffff`.
2. Subprocess Number - the subprocess number of the subprocess in which the exception occured.
3. Fault Identifier - a code which identifies the type of exception which occured (see table 1).
4. Processor Error Code - a processor dependent error code for non-floating point operations.
5. Floating Point Error Code - a processor dependent error code for floating point operations.

The error codes are processor dependent and are only returned where relevant to the cause of the exception.

`0x3333ffff`	Subprocess Number	Fault Identifier	Processor Err Code	FP Err Code

Figure 9: Structure of the Failure Message

Mnemonic	Description	Value
FPFAULT	Floating Point Fault	101
OPFAULT	Opcode Fault	102
ADDRSFAULT	Address Fault	103
DBFAULT	Debug Fault	104
ALIGNFAULT	Alignment Fault	105

Table 1: Error Identifier Values

11 System calls

All system calls implemented within the **Walnut Kernel** use the parameter block to contain all the parameters of the call. There is only one parameter block per process. To prevent subprocesses from altering the parameter block while another subprocess is setting up or receiving the results of a system call it is essential that the reserve field be set to a non-zero value while a subprocess manipulates the parameter block. Setting the reserve field to a value prevents any other subprocess of a process being run until the reserve field is cleared.

11.1 Procedure

How to make a system call:

- Put the call number in the parameter block's reserve field
- Fill in necessary parameters
- Call **system_call()**

After a successful system call has been completed:

- Copy any desired information out of the parameter block
- Set the reserve field to zero

After an unsuccessful system call ($error > 0$)

- Copy the error code and any other desired information out of the parameter block
- Set the reserve field to zero

11.2 Available System Calls

This section describes the currently available system calls on the **Walnut Kernel** and the parameters required for those calls.

11.2.1 Make Object

Name	Symbol	Value
Make Object	**K_MAKEOBJ**	**1**

Input Parameters:

vol	-Volume on which to create object
srights	-System rights
urights	-User rights
limit	-Highest byte offset of object (hard limit)
money	-Initial money
type	-Object type
maxoff	-Highest byte offset of object (soft limit)
maxsz	-Maximum size of object
maxcap	-Maximum number of capabilities including master

Output Parameters:

vol	-Master capability (volume)
serial	-Master capability (serial)
pass1	-Master capability (password 1)
pass2	-Master capability (password 2)
srights	-Master capability (system rights)
urights	-Master capability (user rights)
limit	-Highest byte offset of object (hard limit)
money	-Initial money
type	-Object type
maxoff	-Highest byte offset of object (soft limit)
maxsz	-Maximum size of object
maxcap	-Maximum number of capabilities including master

Description:

This call creates an object of the size specified on the volume specified. The object will have the rights dictated by the **srights** & **urights** field.

Before using the limit value, it is transformed:

$$limit = \begin{cases} BIGLIMIT & \text{if } limit = 0 \\ \text{limit} & \text{otherwise} \end{cases}$$

To create a new object the following preconditions must be met $limit \& 0x3ff = 0$, $maxoff \leq limit$, $limit \leq BIGLIMIT$, and $maxsz \leq BIGLIMIT$.

11.2.2 Derive Capability

Name	Symbol	Value
Derive Capability	**K_MAKECAP**	**2**

Input Parameters:

vol	-Volume
serial	-Serial
pass1	-Password 1
pass2	-Password 2
srights	-System rights mask
urights	-User rights mask
base	-Offset from the beginning of existing view
limit	-Size of derived view
money	-Drawing limit of capability
subpn	-New password 1 (if *subpn* >= 1024)
cindex	-New password 2 (if *subpn* >= 1024)

Output Parameters:

vol	-Volume
serial	-Serial
pass1	-Derived capabilities password 1
pass2	-Derived capabilities password 2
srights	-Derived capabilities system rights
urights	-Derived capabilities user rights
base	-Cleared by call
limit	-Maximum size of derived view
money	-Drawing limit of capability
type	-Drawing limit of derived capability

Description:

This capability derives a capability from a given capability. The new capability may have weaker rights and/or a smaller view of an object. Note that the suicide right may be added to a derived capability.

Attempts to derive capabilities from a capability without the SRMUTLILOAD right always have the same pass2 as the original capability.

If **limit** is set to 0 then the view of the derived capability will extend from the **base** to the end of the view provided by the original capability.

The following pre-conditions must be met $view.limit \geq base$ and $limit \geq 0$.

11.2.3 Delete Capability

Name	Symbol	Value
Delete Capability	**K_DEL**	**3**

Input Parameters:

vol	-Volume
serial	-Serial
pass1	-Password 1
pass2	-Password 2

Output Parameters:

Description:

Deletes the capability specified (if the capability has suicide right) and all of its derivatives (if the capability has derive right).

11.2.4 Delete Derived Capabilities

Name	Symbol	Value
Delete Derived Capabilities	**K_DELDER**	**4**

Input Parameters:

vol	-Volume
serial	-Serial
pass1	-Password 1
pass2	-Password 2

Output Parameters:

Description:

Deletes all of the derivatives of the specified capability (if the capability has derive right).

11.2.5 Resize Object

Name	Symbol	Value
Resize Object	**K_RESIZE**	**5**

Input Parameters:

vol	-Volume
serial	-Serial
pass1	-Password 1
pass2	-Password 2
limit	-New limit
maxoff	-New maximum offset
maxsz	-New maximum size
maxcap	-New maximum number of capabilities

Output Parameters:

Description:

Resizes an object to the values given in **limit**, **maxoff** and **maxsz**. If **maxcap** is greater than the current number of permitted capabilities then the number of capabilities is increased, otherwise, **maxcap** is ignored.

Preconditions: **to be specified**

11.2.6 Shrink Object

Name	Symbol	Value
Shrink Object	**K_SHRINK**	**6**

Input Parameters:

vol	-Volume
serial	-Serial
pass1	-Password 1
pass2	-Password 2

Output Parameters:

Description:

Shrinks the object to a size just sufficient to contain its current contents and sets the limits to make this the maximum size of the object. The object's **limit**, **maximum offset**, **maximum size** and **maximum number of capabilities** are altered.

Preconditions: **to be specified**

11.2.7 Wait

Name	Symbol	Value
Wait	**K_WAIT**	**7**

Input Parameters:

clocktime -Wakeup time

Output Parameters:

Description:

Provided there are no outstanding messages this call puts the subprocess to sleep until either a message arrives or the wakeup time has been reached. The wakeup times of **0** and **-1** have special meanings:

0	Surrender the remainder of time slice
-1	Set no wakeup time. Awake only when sent a message

Wakeup times are in seconds and are absolute. Relative wakeup times can be created by adding a value to the time found in **clocktime**.

11.2.8 Load Capability

Name	Symbol	Value
Load Capability	**K_LOADCAP**	**8**

Input Parameters:

vol	-Volume
serial	-Serial
pass1	-Password 1
pass2	-Password 2
base	-Offset from start of view
limit	-Size of window to be loaded
offset	-Logical address of load location
cindex	-Capability index

Output Parameters:

vol	-Volume
serial	-Serial
pass1	-Password 1
pass2	-Password 2
srights	-System rights of capability
urights	-User rights of capability
base	-Offset from start of view
limit	-Size of window loaded
money	-Drawing right or money provided by capability
offset	-Logical address of load location
cindex	-Capability index

Description:

Loads a view or part of view provided by a capability into the processes address space.

To nominate the capability index of the loaded capability a non-zero **cindex** should be provided to an empty slot in the table of loaded capabilities. If **cindex** is zero then a value will be automatically allocated.

The kernel can be requested to load a capability at a suitable address to contain the view of the object. The following table gives the values of **offset** and their meanings.

0	load anywhere, preferably a large window
1	load anywhere, preferably a small window
2	load as a large window
3	load as a small window

All other values of **offset** are interpreted as specific addresses. The value of offset is truncated to give a page boundary for small windows or a segment boundary for large windows.

Limit gives the size of the window to be loaded. A **limit** of zero specifies that the limit specified by the capability should be used.

11.2.9 Unload Capability

Name	Symbol	Value
Unload Capability	**K_UNLOADCAP**	**9**

Input Parameters:

vol	-Volume
serial	-Serial
pass1	-Password 1
pass2	-Password 2
offset	-Offset of window to be unloaded
cindex	-Index in table of load capabilities of capability to be unloaded

Output Parameters:

limit	-Limit of freed window
offset	-Offset of freed window
cindex	-Index of freed window

Description:

Unloads a capability from address space of the process. If **offset = 0** then the capability **vol serial pass1 pass2** will be unloaded. If **offset = 1** then the capability located at index **cindex** in the table of loaded capabilities will be unloaded. Otherwise the capability at the location **offset** will be unloaded.

11.2.10 Identify Capability

Name	Symbol	Value
Identify Capability	**K_CAPID**	**10**

Input Parameters:

vol	-Volume
serial	-Serial
pass1	-Password 1
pass2	-Password 2
offset	-Offset
cindex	-Index in table of load capabilities

Output Parameters:

vol	-Volume of loaded capability
serial	-Serial of loaded capability
pass1	-Password 1 of loaded capability
pass2	-Password 2 of loaded capability
srights	-System rights of loaded capability
urights	-User rights of loaded capability
limit	-Limit of loaded capability
offset	-Offset of loaded capability
cindex	-Index of loaded capability

Description:

Fills in the rights, **limit**, **offset** and **cindex** for a loaded capability. If **offset = 0** then information for the capability **vol serial pass1 pass2** will be returned. If **offset = 1** then the information for the capability located at index **cindex** in the table of loaded capabilities will be returned. Otherwise information for the capability loaded at location **offset** will be returned.

11.2.11 Make Process

Name	Symbol	Value
Make Process	**K_MAKEPROC**	**11**

Input Parameters:

vol	-Volume to create new process on
srights	-Encoded process parameters
urights	-User rights of new process
base	-Start up time for new process
limit	-Highest byte offset of object (hard limit)
money	-Money to be transferred to new process
type	-Type of new process
maxoff	-Maximum offset of new process object (soft limit)
maxsz	-Maximum size of new process object (soft limit)
maxcap	-Maximum number of capabilities for new process (soft limit)
offset	-Offset at which to load new process object
cindex	-Index in table of loaded capabilities for new process object

Output Parameters:

vol	-Master capability (volume)
serial	-Master capability (serial)
pass1	-Master capability (password 1)
pass2	-Master capability (password 2)
urights	-User rights of new process
limit	-Limit of new process object
money	-Money deposited in new process
type	-Type of new process
maxoff	-Maximum offset of new process object
maxsz	-Maximum size of new process object
maxcap	-Maximum number of capabilities for new process
offset	-Offset of new process object
cindex	-Index of new process object in table of loaded capabilities

Description:

Make Process creates an object, loads the object into the current process's address space and fills in the process state information for the new process.

Initially this call creates an object of the size specified on the volume specified with user rights dictated by the **urights** field and system rights set to SRPROCESSMASTER.

Before using the limit value, it is transformed:

$$limit = \begin{cases} BIGLIMIT & \text{if } limit = 0 \\ \text{limit} & \text{otherwise} \end{cases}$$

The new object is created if the following preconditions are met $limit \& 0x3ff \neq 0$, $maxoff \leq limit$, $limit \leq BIGLIMIT$, and $maxsz \leq BIGLIMIT$.

The object is then loaded into the process's address space at either a nominated location or an automatically allocated location. The location is determined by the value of **offset**. If **offset** is either 0 or 2 then the kernel will allocate a suitable large window automatically and load the object at that location, otherwise the object will be loaded at the segment boundary specified in **offset**.

The capability index of the loaded capability may be nominated by specifying a **cindex** for to an empty slot in the table of loaded capabilities. If **cindex** is zero then a value will be automatically allocated.

A process is then created with the parameters dictated by the **srights** field. The **srights** field is interpreted as four fields of 8 bits:

8 *bits*	8 *bits*	8 *bits*	8 *bits*
Max subp	# message slots	Max loaded caps	# auto load caps

msb lsb

- Max subp - The maximum number of subprocesses for the new process including subprocess 0.
- # message slots - The number of message slots for the new process. As a message slot is reserved for subprocess 0 the number of message slots must be 1 or greater.
- Max loaded caps - The maximum number of loaded capabilities for the new process. This number includes the capability for the process.
- # auto load caps - The number of capabilities to be automatically loaded into the new process's address space including the capability for the new process.

The first four words of the message area contain the initial values of the program counter and stack pointer for subprocess 1. The values are encoded:

message area	index for PC
	initial PC
message area + 2	index for SP
	initial SP

The *index* is the index of a capability in the table of loaded capabilities. If an index value of zero is supplied the initial values are treated as logical addresses instead of as a byte offset from the start of a capability.

The next four words contain the capability of the new process's 'heir'. The heir is notified in case of the death of the process. The message sent contains the remaining cash. If this field contains zero then the master capability for the creating process is used as the heir.

The remainder of the parameter page contains a list of capabilities to be preloaded into the new process's address space. The list is composed of records of the form:

vol	Volume
serial	Serial
pass1	Password 1
pass2	Password 2
base	Start of the loaded window relative to the capability
limit	Size of the loaded window. Zero indicates capability limit
offset	Location of window in the new process's address space
cindex	Index in table of loaded capabilities. Zero for automatic allocation

The creating process will have twice the value indicated in **money** deducted from its cash. This money will be transferred equally to the new process's cash and new process's process object.

The process is scheduled to wake up at the time given in **base** with the wakeup times of **0** and **-1** having the special meanings:

0	Wake up immediately
-1	Set no wakeup time. Awake only when sent a message

Information relating to the new process object is returned to the creating process.

11.2.12 Send Message

Name	Symbol	Value
Send Message	**K_SEND**	**12**

Input Parameters:

vol	-Volume
serial	-Serial
pass1	-Password 1
pass2	-Password 2
money	-Amount of money to be sent to process
limit	-Size of message in bytes
offset	-Offset
subpn	-Subprocess number to send message to
cindex	-Index in table of load capabilities

Output Parameters:

srights	-System rights of loaded capability
urights	-User rights of loaded capability
money	-Amount of money sent to process
limit	-Size of message in bytes
offset	-Offset of loaded capability
cindex	-Index of loaded capability

Description:

Sends a message to a process which is loaded into the address space of the sender. If **offset = 0** then the message will be sent to **vol serial pass1 pass2** provided process object is loaded into the sender's address space. If **offset = 1** then the message will be sent to the process with its process object loaded at index **cindex** in the table of loaded capabilities. Otherwise the message will be sent to the process with its process object loaded at location **offset**. The message length is specified in **limit** in bytes. The message to be sent is located at the beginning of the message area. A positive amount of money - **money** - is removed from sender's cash and sent with the message.

11.2.13 Receive Message

Name	Symbol	Value
Receive Message	**K_RECV**	**13**

Input Parameters:

limit	-Size of match string

Output Parameters:

money	-Amount of money received with message
limit	-Size of message

Description:

Recovers message from a subprocess's message queue. If **limit** is non-zero then only a message which matches the first **limit** characters found in the match string will be recovered. The match string is found at the beginning of the message area. The message received is placed into the message area.

If no message is present an error code is returned

11.2.14 External Send Message

Name	Symbol	Value
External Send Message	**K_EXTSEND**	**14**

Input Parameters:

vol	-Volume
serial	-Serial
pass1	-Password 1
pass2	-Password 2
money	-Amount of money to be sent to process
limit	-Size of message in bytes
subpn	-Subprocess number to send message to

Output Parameters:

money	-Amount of money sent to process
limit	-Size of message in bytes

Description:

Sends a message to the process **vol serial pass1 pass2**. The message length is specified in **limit** in bytes. The message to be sent is located at the beginning of the message area. A positive amount of money - **money** - is removed from the sender's cash and sent with the message.

11.2.15 External Read Memory

Name	Symbol	Value
External Read Memory	**K_EXTREAD**	**15**

Input Parameters:

vol	-Volume
serial	-Serial
pass1	-Password 1
pass2	-Password 2
limit	-Number of bytes to be read
offset	-Offset in bytes from start of capability

Output Parameters:

vol	-Volume
serial	-Serial
pass1	-Password 1
pass2	-Password 2
limit	-Number of bytes read
offset	-Offset in bytes from start of capability

Description:

Reads **limit** bytes from offset **offset** in capability **vol serial pass1 pass2**. The bytes read are stored at the start of the message area.

11.2.16 External Write Memory

Name	Symbol	Value
External Write Memory	**K_EXTWRITE**	**16**

Input Parameters:

vol	-Volume
serial	-Serial
pass1	-Password 1
pass2	-Password 2
limit	-Number of bytes to be written
offset	-Offset in bytes from start of capability

Output Parameters:

vol	-Volume
serial	-Serial
pass1	-Password 1
pass2	-Password 2
limit	-Number of bytes written
offset	-Offset in bytes from start of capability

Description:

Writes **limit** bytes from offset **offset** in capability **vol serial pass1 pass2**. The bytes to be written are stored at the start of the message area.

11.2.17 Bank

Name	Symbol	Value
Bank	**K_BANK**	**17**

Input Parameters:

vol	-Volume
serial	-Serial
pass1	-Password 1
pass2	-Password 2
money	-Amount of money to be transferred from capability to cash

Output Parameters:

vol	-Volume
serial	-Serial
pass1	-Password 1
pass2	-Password 2
srights	-System rights of capability
urights	-User rights of capability
limit	-Size of view in bytes
money	-Drawing limit available to capability

Description:

Transfers **money** from cash from the calling process to the capability **vol serial pass1 pass2**. Both positive and negative amounts of cash may be transferred.

If **money** is positive then the capability must have deposit right to perform the transfer. If **money** is negative then the capability must have withdraw right to perform the transfer.

11.2.18 Restrict Rights

Name	Symbol	Value
Restrict Rights	**K_RESTRICT**	**18**

Input Parameters:

vol	-Volume
serial	-Serial
pass1	-Password 1
pass2	-Password 2
srights	-System rights mask
urights	-User rights mask

Output Parameters:

vol	-Volume
serial	-Serial
pass1	-Password 1
pass2	-Password 2
srights	-System rights of capability
urights	-User rights of capability

Description:

Reduces the rights of a capability by performing a bitwise and of the rights masks supplied with the rights bitmaps of the capability **vol serial pass1 pass2**.

The capability named must have suicide right for restrict to operate.

11.2.19 Capability Status

Name	Symbol	Value
Capability Status	**K_CAPSTAT**	**19**

Input Parameters:

vol	-Volume
serial	-Serial
pass1	-Password 1
pass2	-Password 2

Output Parameters:

vol	-Volume
serial	-Serial
pass1	-Password 1
pass2	-Password 2
srights	-System rights of capability
urights	-User rights of capability
base	-Cleared by call
limit	-Limit of view of capability
money	-Withdrawal right of capability
type	-Type of object
maxoff	-Maximum offset of object
maxsz	-Maximum size of object
maxcap	-Maximum number of capabilities for object

Description:

Returns details of capability **vol serial pass1 pass2** and associated object.

11.2.20 Rename Capability

Name	Symbol	Value
Rename Capability	**K_RENAME**	**20**

Input Parameters:

vol	-Volume
serial	-Serial
pass1	-Password 1
pass2	-Password 2

Output Parameters:

vol	-Volume
serial	-Serial
pass1	-Password 1
pass2	-Password 2
base	-Cleared by call

Description:

Changes the passwords of capability **vol serial pass1 pass2** to a new pair of random values.

A precondition to this call is that the capability has suicide right. In addition the master capability of a process cannot be renamed.

11.2.21 Make Subprocess

Name	Symbol	Value
Make Subprocess	**K_MAKESUBP**	**21**

Input Parameters:

base	-Start up time for new subprocess
limit	-Priority of new subprocess
subpn	-Subprocess number

Output Parameters:

base	-Start up time for new subprocess
limit	-Priority of new subprocess
subpn	-Subprocess number of new subprocess

Description:

Creates a new subprocess of the current process. If **subpn** is not zero and no subprocess of the current process has been allocated that number then the subprocess's number will be **subpn**. The priority is set to the least 8 bits of **limit**. The subprocess is scheduled to wake up at the time given in **base** with the wakeup times of **0** and **-1** having the special meanings:

0	Wake up immediately
-1	Set no wakeup time. Awake only when sent a message

The first four words of the message area contain the initial values of the program counter and stack pointer for the new subprocess. The values are encoded:

message area	index for PC
	initial PC
message area + 2	index for SP
	initial SP

The *index* is the index of a capability in the table of loaded capabilities. If an index value of zero is supplied the initial values are treated as logical addresses instead of as a byte offset from the start of a capability.

11.2.22 Delete Subprocess

Name	Symbol	Value
Delete Subprocess	**K_DELSUBP**	**22**

Input Parameters:

subpn	-Subprocess number

Output Parameters:

subpn	-Subprocess number of deleted subprocess

Description:

Deletes subprocess **subpn**. Note that neither subprocess 0 nor 1 can be deleted.

11.2.23 Load Register Set

Name	Symbol	Value
Load Register Set	**K_LOADREG**	**23**

Input Parameters:

subpn	-Subprocess number

Output Parameters:

subpn	-Subprocess number

Description:

Copies the structure sysstate at the start of the message area into subprocess table entry **subpn**.

11.2.24 Save Register Set

Name	Symbol	Value
Save Register Set	**K_SAVEREG**	**24**

Input Parameters:

subpn	-Subprocess number

Output Parameters:

subpn	-Subprocess number

Description:

Copies the structure sysstate from subprocess table entry **subpn** into the start of the message area.

11.2.25 Set Trap

Name	Symbol	Value
Set Trap	**K_SETTRAP**	**25**

Input Parameters:

offset	-Subprocess number to send trap message to
subpn	-Subprocess number whose trap is being set

Output Parameters:

offset	-Subprocess number to send trap message to
subpn	-Subprocess number whose trap is being set

Description:

Sets the destination subprocess for trap messages. Subprocess **offset** is notified of faults in subprocess **subpn**.

11.2.26 Receive Message and Close Box

Name	Symbol	Value
Receive Message Close	**K_RECV_CLOSE**	**26**

Input Parameters:

limit	-Size of match string

Output Parameters:

money	-Amount of money received with message
limit	-Size of message

Description:

Recovers message from a subprocess's message queue and closes the mail box the message is recovered from. If **limit** is non-zero then only a message which matches the first **limit** characters found in the match string will be recovered. The match string is found at the beginning of the message area. The message received is placed into the message area.

If no message is present an error code is returned.

11.2.27 Accept Mail

Name	Symbol	Value
Accept Mail	**K_ACCEPT_MAIL**	**27**

Input Parameters:

limit	-Size of match string
subpn	-subprocess for which mail box is reserved

Output Parameters:

Description:

Opens a mail box for a subprocess and sets the acceptance string for the mail box. The mail box is taken from the pool of closed mail boxes and set to receive messages for a specific subprocess **subpn** or if **subpn** is `0xFF` the mail box can be used for any subprocess.

If **limit** is non-zero then the mail box created will only accept messages which match the first **limit** characters found in the match string when the mail box is opened. The match string is found at the beginning of the message area.

11.2.28 Close Mail Box

Name	Symbol	Value
Close Matching Mail Boxes	**K_CLOSE_BOX**	**28**

Input Parameters:

limit	-Size of match string
subpn	-subprocess for which mail box is reserved

Output Parameters:

base	-Number of mail boxes closed by operation

Description:

Closes mail boxes which match the closing criteria. If **subpn** equals `0xFF` and **limit** is zero then all user mail boxes will be closed. If **limit** is non-zero then only user mail boxes with match strings matching the first **limit** characters of the match string found at the beginning of the message area will be closed. If **subpn** is non-zero then only user mail boxes for subprocess **subpn** are closed.

11.2.29 Copy Object

Name	Symbol	Value
Copy Object	**K_COPYOBJ**	**29**

Input Parameters:

vol	-Volume (original)
serial	-Serial (original)
pass1	-Password 1 (original)
pass2	-Password 2 (original)
srights	-System rights mask
urights	-User rights mask
base	-Start of copy relative to beginning of original
limit	-End of copy relative to base
money	-Money to be transferred to copy
type	-Type of copy
maxsz	-Maximum size of copy
maxcap	-Maximum number of capabilities of copy

Output Parameters:

vol	-Volume (copy)
serial	-Serial (copy)
pass1	-Password 1 (copy)
pass2	-Password 2 (copy)
srights	-System rights of copy
maxoff	-Maximum offset of copy
maxcap	-Maximum number of capabilities of copy

Description:

Duplicates an object by creating a new object and copying the contents of the original object to the new object. This call copies only the defined pages of an object and hence produces an exact duplicate of the contents of the section of the object referred to by the capability for the original object. The rights fields allow the rights of the copy to be reduced as the rights mask and the rights fields are combined by a bitwise AND to produce the copy's rights field. The **money** field indicates the amount of money to be transferred from the process cash to the new object. The **maxsz** field specifies the maximum size of the new object. The **type** field specifies the type of the copy. The **base** field specifies the start of the the copy region which extends through to **limit**. If the **limit** and **base** fields are zero then the complete object is copied.

NOTE:

- This call will **not** duplicate processes
- This call corrupts the first four words of the message area

11.2.30 Check Process State

Name	Symbol	Value
Peek Process	**K_PEEK_PROC**	**30**

Input Parameters:

vol	-Volume
serial	-Serial
pass1	-Password 1
pass2	-Password 2

Output Parameters:

srights	-State of process
base	-Wakeup time

Description:

Returns the state and wakeup time of a process given a suitable capability (capability must have SRPEEK right). for the process. The wakeup time is returned in **base** and the process state in **srights**. The process state is encoded:

Value	State
-2	No such process
-3	No right to inquire
1	Process normal
2	Process in kernel
3	Process in read fault
4	Process in write fault
5	Process frozen
6	Process in probate
7	Process dead

11.2.31 Set Heir of Process

Name	Symbol	Value
Set Heir	**K_SET_HEIR**	**31**

Input Parameters:

vol	-Volume of heir
serial	-Serial of heir
pass1	-Password 1 of heir
pass2	-Password 2 of heir

Output Parameters:

Description:

Set the heir of a process to the capability **vol serial pass1 pass2**. The heir of a process receives a process's death message and any remaining cash.