apeNEXT OS1

31 August 2005

This document describes the initial version of the apeNEXT operating system, called OS1. It will support only a limited number of system services and is likely to only use the I2C links between frontend PC and apeNEXT. This simple operating system does not (yet) foresee data transport routines to run on the apeNEXT nodes. The operations executed on the frontend PC are handled by a modified version of nose.

Contents

List of Tables

1 Supported system services

The following system services will be supported:

Table 1: *System services.*

Bootstrap (BST)	All operations necessary before putting machine into run-mode, except program loading
Memory load (MLD)	Load program and data into main memory
Start (STRT)	Start program execution
Exit (EXIT)	Handle end of program execution
Slice write (SWR)	Data write by all nodes of specified slice
Global write (GWR)	Global data (e.g. variables assumed to be global) write by one particular node
Slice read (MRD)	Multidata read by all nodes of specified slice
Broadcast read (BRD)	Broadcast read by all nodes of specified slice
File open (FOP)	Opening a file
File close (FCL)	Closing a file
Time (TIM)	Read time since the Epoch
File seek (FSE)	Reposition file
Set random seed (SRA)	Set random number generator seed
Read random numbers (MRA/BRA)	Read from random number generator (multi-data/broadcast)
C-global-printf (PRF)	Formatted (global) write using format string
C-slice-printf (SPRF)	Formatted slice write using format string
C-broadcast-scanf (SCF)	Formatted read using format string
C-multidata-scanf (MSCF)	Formatted multidata read using format string
C command-line args (MSCF)	Write command line arguments to memory

2 Low-level protocol (= interface apeNEXT–frontend PC)

Most of the system services defined in the previous section require a low-level protocol for interaction between a particular node of apeNEXT (called SYS_NODE) and the frontend PC. The relevant information is exchanged via an

OS operation request (os_req): The operation code is written by the machine into bits 37:32 of configuration register CrMemExc (0x20). An overview on all operations is given in Tab. 2.

Table 2: *OS operation request codes.*

0x00	SYS_REQ_NOP	no operation
0x01	SYS_REQ_EXIT	EXIT
0x02	SYS_REQ_IO	SWR, GWR, MRD, BRD, FOP, FCL, TIM, PRF, SCF, MRA, BRA

Table 2: *OS operation request codes.*

0x03	SYS_REQ_FSE	FSE
0x04	SYS_REQ_SET	SRA
0x05	SYS_REQ_PARGS	PARGS
0x20	SYS_REQ_OTHER	

Return value flags (os_rvalflg): The return value flags are written by the machine into bits 39:38 of configuration register CrMemExc (0x20). An overview on all operations is given in Tab. 3.

Table 3: *Return value flags.*

0x00	SYS_RVAL_NOP	no operation
0x40	SYS_RVAL_GET	request return value
0x80	SYS_RVAL_RST	reset return value

OS argument (os_arg): This argument is written into a reserved 128 bit register. The encoding depends on the OS request.

Exit: In case of an regular program exit (EXIT) the os_arg contains the user defined exit code.

I/O: In case of I/O operations the os_arg contains the information list in Tab. 4. The encoding of the argument is defined in Tab. 12. Not all fields of the argument will be relevant for a given I/O operation. Those fields can in this case contain arbitrary values.[1]

Table 4: *I/O operation arguments.*

address (addr)	Memory or register file address from/to which data is to be read/written.
bank selector (bs)	Selects which part of the 128 bit will be written.
command (cmd)	Type of operation as listed in Tab. 6.
file descriptor (fd)	Small, non-negative integer which identifies the file from/to which data is read/written. A list of reserved file descriptors is listed in Tab. 7.
format (fmt)	Format of the data to be written/read. A list of all formats is listed in Tab. 8.
bank separator (bsep)	Defines token which is written between high and low part of a word. It is only used if bs=SYS_IOBS_V and fmt!=SYS_IOFMT_BIN. A list of all separators is listed in Tab. 9.

[1] E.g., in case of a global write (GWR) the definition of a slice is irrelevant.

Table 4: *I/O operation arguments.*

word separator (`wsep`)	Defines token which is written between different words. It is only used if `fmt`!=`SYS_IOFMT_BIN`. It is not printed after writing the last word. A list of all separators is listed in Tab. 9.
packet separator (`psep`)	Defines token which is written at the end of the packet, irrespective of whether there is a next packet. It is only used if `fmt`!=`SYS_IOFMT_BIN`. A list of all separators is listed in Tab. 9.
length (`len`)	Number of words to be read or written. If `len`>0xffff packets will be split without changing any of the other fields.
slice (`slice`)	Since this version of the operating system is tailored for the initial hardware tests, it is safe to assume that the maximum machine topology is $8 \times 8 \times 8$ (i.e. one rack). Any slice $[k_x, k_y, k_z][l_x, l_y, l_z]$ can therefore be encoded in $2\times(8+4+4) = 32$ bits. The special value `SYS_IOSLICE_ALL` is used to select all nodes of the machine. Value ignored for `cmd`=`SYS_IOCMD_GWR`.
device (`dev`)	Defines the device from/to which data is to be read/written. Currently there are two devices: memory and register file. In case of a read operation to the register file, `bs`=`SYS_IOBS_V` is mandatory.
read flags (`rdflg`)	Defines flags needed for initialization of high bank during read operations as defined in Tab. 11.
file open mode (`mode`)	Modes used for file open operation as listed in Tab. 10. Ignored when `cmd`!=`SYS_IOCMD_FOP`.
big I/O flag (`bigflg`)	If this flag is set the total length cannot be encoded in the OS1 header and will be read from a different register.
counter (`cnt`)	Counts the number of packets in case of composed services. Counter will be decremented for consecutive operations.
version (`version`)	Version number of the protocol.

File seek: In case of the `FSE` service the `os_arg` will encode the following values: file descriptor (`fd`), offset (`offset`), position specifier (`whence`). The position specifier can have one of the following values: `SYS_FSE_SET`, `SYS_FSE_CUR`, `SYS_FSE_END`. The encoding of the argument is defined in Tab. 5.

Table 5: *Argument encoding service FSE.*

`fd`	7:0	`0000 0000 0000 0000 0000 0000 0000 00FF`
`whence`	9:8	`0000 0000 0000 0000 0000 0000 0000 0300`
`offset`	94:64	`FFFF FFFF FFFF FFFF 0000 0000 0000 0000`

Set random seed: In case of the `SRA` service the `os_arg` will contain the adress where to load the new seed for the random number generator.

OS return value (`os_ret`): If a return value is requested after performing a system service it is written to the low bank of `REG_SYS_RVAL`. In case of an error an error number is written into the high bank of that register. Typically the error number by the corresponding system operation executed on the front-end system will be return.[2] If the return value reset flag is set, the front-end system will reset the variables storing return values before performing the next system service.

Table 6: *List of I/O command codes.*

0x00	`SYS_IOCMD_NOP`
0x01	`SYS_IOCMD_SWR`
0x02	`SYS_IOCMD_GWR`
0x03	`SYS_IOCMD_MRD`
0x04	`SYS_IOCMD_BRD`
0x05	`SYS_IOCMD_FOP`
0x06	`SYS_IOCMD_FCL`
0x07	`SYS_IOCMD_TIM`
0x09	`SYS_IOCMD_PRF`
0x0a	`SYS_IOCMD_SCF`
0x0b	`SYS_IOCMD_MRA`
0x0c	`SYS_IOCMD_BRA`
0x0d	`SYS_IOCMD_SPRF`
0x0e	`SYS_IOCMD_MSCF`

Table 7: *List of reserved file descriptors.*

0	stdin
1	stdout
2	stderr

Table 8: *List of formats and the corresponding standard C formats (last column).*

0x0	`SYS_IOFMT_BIN`	binary (=unformatted)	
0x1	`SYS_IOFMT_INT`	integer	`%d`
0x2	`SYS_IOFMT_UINT`	unsigned integer	`%u`
0x3	`SYS_IOFMT_HEX`	hexadecimal integer	`0x%x`
0x4	`SYS_IOFMT_STR`	string	`%s`
0x5	`SYS_IOFMT_VARSTR`	variable string	`%s`

[2]See, e.g., on Linux systems: `/usr/include/asm/errno.h`.

Table 8: *List of formats and the corresponding standard C formats (last column).*

0x6	SYS_IOFMT_DBL	double	%e

Table 9: *List of separators. The lower set of "separators" can only be used as bank separators.*

0x00	SYS_IO[BWP]SEP_NOP	”
0x01	SYS_IO[BWP]SEP_SPACE	’ ’
0x02	SYS_IO[BWP]SEP_TAB	\t
0x03	SYS_IO[BWP]SEP_NL	\n
0x04	SYS_IO[BWP]SEP_KOMMA	,
0x08	SYS_IOBSEP_RBRACKET	(#,#)
0x09	SYS_IOBSEP_SBRACKET	[#,#]

Table 10: *List of file open modes with the corresponding argument used by `fopen()`.*

0x1	SYS_IOMODE_R	r
0x2	SYS_IOMODE_RPLUS	r+
0x3	SYS_IOMODE_W	w
0x4	SYS_IOMODE_WPLUS	w+
0x5	SYS_IOMODE_A	a
0x6	SYS_IOMODE_APLUS	a+

Table 11: *List of read flags.*

0x0	SYS_IORDFLG_NOP	high bank unchanged
0x1	SYS_IORDFLG_HZERO	initialize high bank with zero
0x2	SYS_IORDFLG_HMIRROR	mirror content low bank to high bank

Table 12: *I/O argument encoding.*

addr	25:0	0000 0000 0000 0000 0000 0000 03FF FFFF
bs	29:28	0000 0000 0000 0000 0000 0000 3000 0000
cmd	39:32	0000 0000 0000 0000 0000 00FF 0000 0000
fd	47:40	0000 0000 0000 0000 0000 FF00 0000 0000
fmt	51:48	0000 0000 0000 0000 000F 0000 0000 0000
bsep	55:52	0000 0000 0000 0000 00F0 0000 0000 0000
wsep	59:56	0000 0000 0000 0000 0F00 0000 0000 0000
psep	63:60	0000 0000 0000 0000 F000 0000 0000 0000
len	79:64	0000 0000 0000 FFFF 0000 0000 0000 0000
slice	111:80	0000 FFFF FFFF 0000 0000 0000 0000 0000

Table 12: *I/O argument encoding.*

dev	113:112	0003 0000 0000 0000 0000 0000 0000 0000
rdflg	115:114	000C 0000 0000 0000 0000 0000 0000 0000
mode	118:116	0070 0000 0000 0000 0000 0000 0000 0000
bigflg	119:119	0080 0000 0000 0000 0000 0000 0000 0000
cnt	123:120	0F00 0000 0000 0000 0000 0000 0000 0000
version	127:124	F000 0000 0000 0000 0000 0000 0000 0000

3 Assembler macros (= interface compiler–OS1)

To make the OS services available on assembler level a set of expape macros is provided, which are listed in Tab. 13. For convenience, e.g. for assembler programming, there are some additional routines as listed in Tab. 14.

The macros are stored in the file `os1-macros.exp` and require the files `nsys.exp` and `os1.exp` to be included first. Furthermore, it is assumed that setting-up the static cache is done elsewhere.

Required system subroutines will be provided by executing the macro `$os1_sub`.

Any set of I/O operation (`os_req` = `SYS_REQ_IO`) is started by the macro `os1_iostart` and ended by `$io_end`. The slice can be defined by the macros `$io_slice()` or `$io_slice_all` and has to be defined only once during a set of I/O operations. Similarly, the file descriptor, which is defined by the macros `$io_fd()` or `$io_fopen()` and undefined by `$io_fclose()`.

Table 13: *Assembler macros.*

`$sys_utilities`	- subroutines for system handling
`$sys_exit(status)` `status` ..... MOP	- initialize `os_arg` register - halt machine using `os_req`
`$io_start`	- setup internal variables - initialize registers
`$io_fd(fd)` `fd` ..... MOP	- setup internal variables - initialize `fd` register
`$io_cnt_set(cnt)` `cnt` ..... REG	- copy given register to `cnt` register - setup internal flag
`$io_cnt_decr`	- decrement counter by one - update `cnt` register
`$io_cnt_clear`	- setup internal flag

Table 13: *Assembler macros.*

`$io_setmode(reg, mode)` `reg` ..... MOP `mode` ..... CONST	initialize register translating generic into OS1 encoding

`$io_fopenp(fd, mode, ptr, dev, len)`
`fd` ..... MOP
`mode` ..... REG (OS1 encoded)
`ptr` ..... MOP
`dev` ..... constant
`len` ..... MOP or `""`
	- initialize `os_arg` register
	- halt machine using `os_req`

`$io_fclose(fd)` `fd` ..... MOP	- initialize `os_arg` register - halt machine using `os_req`
`$io_slice(kx, ky, kz, lx, ly, lz)` `kx,...,lz` ..... MOP	- setup internal variables - save encoded slice in register - set `slice` register
`$io_slice_set`	- set `slice` to stored value
`$io_slice_all`	- set `slice` register `IOSLICE_ALL`

`$io_packet(cmd, fmtsep, rdflg, dev, len, bs, addr)`
`cmd` ..... constant
`fmtsep` ..... constant
`rdflg` ..... constant
`dev` ..... constant
`len` ..... MOP
`bs` ..... constant
`addr` ..... MOP
`fmtsep` = `fmt|bsep|wsep|psep`
	- initialize `os_arg` register
	- halt machine using `os_req`

`$io_returnval(ret)` `ret` ..... REG	- setup internal variables
`$io_end`	- setup internal variables
`$sys_setwhence(reg, whence)` `reg` ..... MOP `whence` ..... CONST	initialize register translating generic into OS1 encoding

Table 13: *Assembler macros.*

`$sys_fseek(fd, offset, whence, ret)` `fd` ..... MOP `offset` ..... MOP `whence` ..... REG (OS1 encoded) `ret` ..... const	- setup internal variables - initialize `os_arg` register - halt machine using `os_req` - optionally handle return value
`$sys_srand(maddr)` `maddr` ..... MOP	- initialize `os_arg` register - halt machine using `os_req` - read value from given address
`$sys_pargss(addr, argc)` `addr` ..... MOP `argc` ..... REG	- initialize `os_arg` register - halt machine using `os_req`

Table 14: *Additional assembler macros.*

`$io_node(kx, ky, kz)` `kx, ky,kz` ..... MOP	Define single I/O node: - `$io_slice(kx,ky,kz,kx,ky,kz)`

`$io_putd(int)`
`$io_rputd(int)`
`$io_putdv(int)`
`$io_rputdv(int)`
`int` ..... MOP
	Write memory/register using format `%d` or `[%d,%d]`:
	- use `io_packet()`

`$io_putx(int)`
`$io_rputx(int)`
`$io_putxv(int)`
`$io_rputxv(int)`
`int` ..... MOP
	Write memory/register using format `%x` or `[%x,%x]`:
	- use `io_packet()`

`$io_pute(int)`
`$io_rpute(int)`
`int` ..... MOP
	Write memory/register using format `%e`:
	- use `io_packet()`

`$io_putz(int)`
`$io_rputz(int)`
`int` ..... MOP
	Write memory/register using format `(%e,%e)`:
	- use `io_packet()`

`$io_puts(str)` `str` ..... constant	Write constant string: - [initialize string] - initialize `os_arg` register - halt machine using `os_req`

Table 14: *Additional assembler macros.*

`$io_getd(int)`
`$io_rgetd(int)`
`$io_getdv(int)`
`$io_rgetdv(int)`
`int` ..... MOP
	Read to register/memory as integer:
	- use `io_packet()`

`$io_gete(int)`
`$io_rgete(int)`
`int` ..... MOP
	Read to register/memory as double:
	- use `io_packet()`

`$io_getz(int)`
`$io_rgetz(int)`
`int` ..... MOP
	Read to register/memory as complex:
	- use `io_packet()`

`$io_fopen(fd, mode, str)` `fd` ..... MOP `mode` ..... constant `str` ..... constant	- initialize string (=file name) - use `$io_fopenp()`

3.1 Example program

Here an example program for writing the `node_abs_id` (stored in memory) to standard output:

```
\include os1.exp

PRAGMA_MPP_REGS $REG_MPP_START $REG_MPP_END
$CACHE_INIT

$io_start
$io_fd($SYS_IOFD_STDOUT)
$io_slice(0, 0, 0, 1, 1, 1)
$io_packet($SYS_IOCMD_SWR,\
     <$SYS_IOFMT_DBL .or. $SYS_IOPSEP_NL>, $SYS_IORDFLG_NOP,\
     $SYS_IODEV_MEM, 1, $SYS_IOBS_L, $MEM_NODE_ABS_ID)
$io_end

$sys_exit(0)

$CACHE_STATIC
$sys_utilities
$CACHE_END
```

3.2 Default values

If not defined the following default values are used:

- Slice defaults to all nodes (equivalent to `$io_slice_all`).
- The file descriptor defaults to `stdin` in case of read and `stdout` in case of write operations.

3.3 Run-time error handling and return values

Whenever an macro initiates a system service, the return value will be copied into a given register. In case of the I/O operations this can be specified by the macro `$io_returnval()`. If no such register number has been defined or if this has been set to -1 instructions will be inserted that initiate program abortion in case of an error.

4 Memory layout

The memory layout is shown in Fig. 1. The following addresses are compile time constants:

Table 15: *Memory layout constants.*

`MEM_SYS_START`	first system memory address
`MEM_SYS_END`	last system memory address
`MEM_SYS_DMSTART`	first system data memory address
`MEM_SYS_DMEND`	last system data memory address
`MEM_USR_START`	first user memory address
`MEM_USR_DMSTART`	first user data memory address

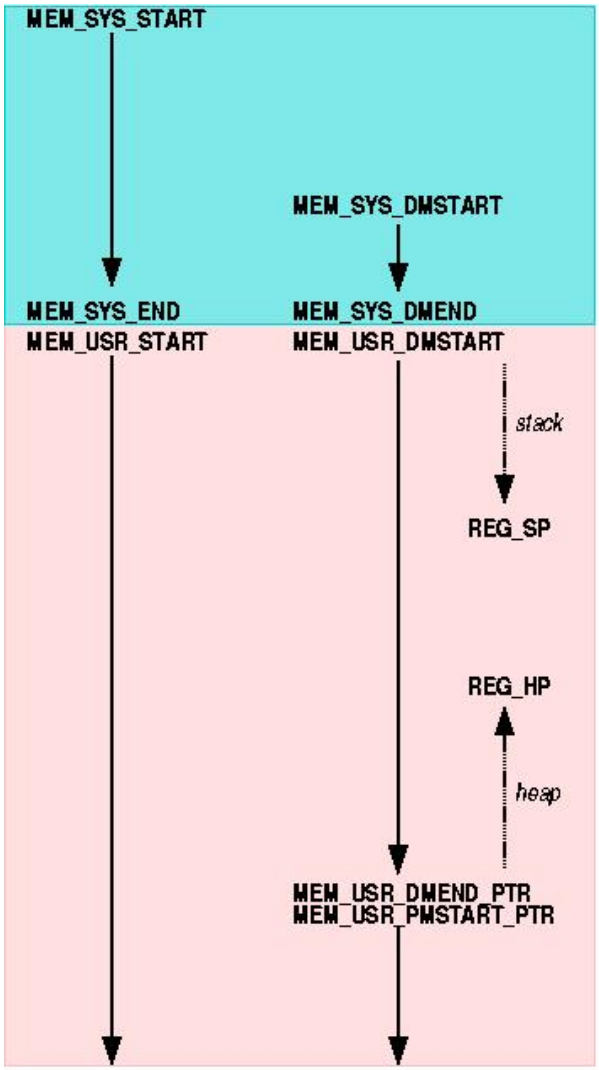

Figure 1: *Memory layout.*

Memory addresses reserved for run-time variables:

Table 16: *Pre-initialized memory addresses.*

Name	Init./Upd.	Description
MEM_MACHINE_SIZE_X MEM_MACHINE_SIZE_Y MEM_MACHINE_SIZE_Z	nose	machine size
MEM_NODE_ABS_ID	nose	processor id
MEM_NODE_ABS_X MEM_NODE_ABS_Y MEM_NODE_ABS_Z	nose	processor coordinates
MEM_USR_DMEND_PTR	nose (linker)	pointer to end of user data memory
MEM_USR_PMSTART_PTR	nose (linker)	pointer to start of user program memory

5 Register file layout

The following register or register ranges will be used by OS1 or other programs, e.g. compilers or mpp:

Table 17: *Register file layout [0..63].*

REG_SYS_ZERO	integer/floating point zero
REG_SYS_IONE	integer one
REG_FP	frame pointer
REG_SP	stack pointer
REG_HP	heap pointer
REG_RET	return address
REG_ARG0	subroutine argument 0
REG_ARG1	subroutine argument 1
REG_ARG2	subroutine argument 2
REG_ARG3	subroutine argument 3
REG_MPP_START	first temporary register used by mpp
REG_MPP_END	last temporary register used by mpp
REG_SYS_ARG	os_arg and os_ret
REG_SYS_ASTART	first temporary register used by OS1
REG_SYS_AEND	last temporary register used by OS1
REG_USR_ASTART	first physical user register
REG_USR_AEND	last physical user register

Table 18: *Register file layout [64..255].*

REG_FONEV	floating point (1.,1.)
REG_FONEL	floating point (1.,0.)

Table 18: *Register file layout [64..255].*

REG_FONEH	floating point (0.,1.)
REG_FTWO	floating point (2.,0.)
REG_SYS_DSTART	first temporary register used by OS1
REG_SYS_DEND	last temporary register used by OS1
REG_USR_DSTART	first physical user register
REG_USR_DEND	last physical user register

Table 19: *Virtual registers.*

REG_SYS_VSTART	first virtual system register
REG_SYS_VEND	last virtual system register
REG_USR_VSTART	first vitual user register

Any of the registers marked as "temporary used by OS1" might be modified during execution of OS1 macros. The registers might be re-used otherwise.

Need convention about who initializes constants!

6 System services details

6.1 Exit (EXIT)

The user defined exit status value will be written to os_arg register. After program execution ended with os_req=SYSREQ_EXIT this status will be read by OS1 and returned to the calling shell.

APE	**frontend**
write user status to os_arg	poll GLB_HALT
halt machine	poll GLB_HALT
	read os_req from SYS_NODE
	read os_arg from SYS_NODE
	exit(user status)

6.2 Slice write (SWR)

If program execution stops with os_req=SYSREQ_IO OS1 will read the os_arg register and decode the argument. If the cmd field in the argument is equal to SYS_IOCMD_SWR then the required number of words will be read from the given address of all nodes which are within the slice.

APE	frontend
initialize os_arg (1st packet)	poll GLB_HALT
halt machine	poll GLB_HALT
	read os_req from SYS_NODE
	read os_arg from SYS_NODE
	read data from nodes in slice
	(formatted) write to given fd
	restart machine
initialize os_arg (2nd packet)	poll GLB_HALT
halt machine	poll GLB_HALT
...	...

6.3 Multidata read (MRD)

Similar to SWR.

6.4 Global write (GWR)

If program execution stops with os_req=GWR OS1 will read the os_arg register and decode the argument. The required number of words will then be read from the given address of one predefined node (with respect to future versions of the OS it should be a node connected with an external HIB).

6.5 Broadcast read (BRD)

If program execution stops with os_req=BRD OS1 will read the os_arg register and decode the argument. At the given address of each node the same data will be written.

6.6 File open (FOP)

Before going into system mode the machine will write the file name in one or more temporary registers. When program execution stops with os_req=FOP OS1 will read the os_arg register and decode the argument. It then reads the file name from the given address and does a open() system call at the frontend system. OS1 has to keep a table which maps the file descriptor used by the machine and the one returned by the open() call on the frontend.

6.7 File close (FCL)

The file descriptor on the frontend system which corresponds to the file descriptor provided by the machine is closed.

6.8 Read time (TIM)

This operation is equivalent to BRD except that the read data contains the integer representation of the time since the Epoch (00:00:00 UTC, January 1, 1970), measured in seconds.

6.9 Seek file position (FSE)

Set file position for the given file descriptor. The new position, measured in bytes, is obtained by adding offset bytes to the position specified by the 'whence' argument.

6.10 Set random number seed (SRA)

Machine provides one initial value to initiate random number generator. Only the low bank is used, irrespective of the of bs.

6.11 Read random numbers (MRA/BRA)

This operation is equivalent to MRD/BRD except that instead from a file descriptor data is taken from a random number generator. If fmt = SYS_IOFMT_DBL floating point numbers in the range $[0.0, 1.0]$ will be returned, otherwise unsigned 64-bit fix point numbers are read to each of the selected banks

6.12 C-printf (PRF, SPRF)

6.13 C-scanf (SCF, MSCF)

6.14 Command line arguments (PARGS)

Via the assemble macro $sys_pargss the machine will provide a memory and a register address to the frontend system. After performing halt using request code SYS_REQ_PARGS the frontend system will load to the memory address

- Number of arguments
- Total number of words stored to memory
- Pointer to first argument
- ...
- Pointer to last argument
- First argument terminated by zero word
- ...
- Last argument terminated by zero word

If succesfull, the low bank of the register will be set to '0' and the high bank to the number of arguments.

7 System services return values and error numbers

For the system services listed below return values and error numbers are foreseen. For all other services nothing is returned, i.e. the `os_ret` register is undefined.

7.1 SWR, MRD, GWR, BRD

Upon success this service returns zero. No error numbers are foreseen.

7.2 FOP, FCL

Upon success a zero value is return.[3] If an error occurs, the error number is set according to the error number used on the front-end system.

7.3 FSE

Upon succesful completion 0 is returned. Otherwise, -1 is returned and the error number is set to indicate the error.

8 I/O Details

8.1 Format

Aim of the format encoding is to define an coding which allows a unique mapping into a standard-C format string. The following fields in the I/O argument are relevant for this: `bs`, `len`, `fmt`, `bsep`, `wsep`, `psep`. The I/O macros will have to take care (at compile time) that this field values are consistent.[4] Examples:

len	bs|fmt	bsep	wsep	psep	format
2	L|DBL	(ignored)	TAB	NL	"%e\t%e\n"
1	V|DBL	KOMMA	(ignored: len=1)	NL	"%e,%e\n"
1	V|DBL	RBRACKET	(ignored: len=1)	NL	"(%e,%e)\n"

8.2 Bank selector and read flags

During read operations the read flags will be used to construct a 128-bit word. Depending on the bank selector setting only the high or low 64-bit part of this word will be written. Since OS1 uses the I2C channel for I/O operations, reading to only one bank (i.e. `bs` $\neq$ `SYS_IOBS_V`) of the register file (i.e. `dev` = `SYS_IODEV_RF`) is currently not supported.

[3] Note that for `FOP` this is not consistent with standard `fopen()` since the file descriptor is generated by the compiler, not OS1.

[4] E.g., `bs==SYS_IOBS_L` and `bsep!=SYS_IOSEP_NOP` is an inconsistent choice.

8.3 Additional macros `$io_get*()` and `$io_put*()`

These additional macros are introduced to simplify assembler programming. The following actions are performed implicitely:

- The statements `$io_start` and `$io_end` may be ommitted.[5]
- If slice is not defined (slice is defined) then the get and put macros will use the `BRD` (`MRD`) and `GWR` (`SWR`) services, respectively.

9 TAO issues

The TAO statements for I/O operations are handled as follows:

Table 20: *TAO write statements vs. OS1 I/O operations.*

TAO write	OS1		`rasm`	comment
	`cmd`	`slice`		
i	`GWR`	ignored	`GWR`	write single global integer (independent of slice)
r	`GWR`	ignored	`GWR`	write single local real (independent of slice)
slice i	`GWR`	ignored	`GWR`	write global integer (independent of slice)[6]
slice r	`SWR`	specified	`SWR`	write local real

Table 21: *TAO read statements vs. OS1 I/O operations.*

TAO write	OS1		`rasm`	comment
	`cmd`	`slice`		
i	`BRD`	default = all	`BRD`	read global integer
r	`BRD`	default = all	`BRD`	read local real (same value on all nodes)
multidata i	`BRD`	default = all	`BRD`	read global integer
multidata r	`MRD`	default = all	`MRD`	read local real (different values on all nodes)
slice i	`BRD`	all	`BRD`	read global integer (set slice to all)
slice r	`BRD`	specified	`BRD`	read local real (same value on all nodes)
slice multidata i	`BRD`	all	`BRD`	read global integer (set slice to all)
slice multidata r	`MRD`	specified	`MRD`	read local real (ignore slice)

10 Miscellanious

Definition strings: Strings are 128-bit aligned (padded by zero bytes if necessary). Strings which length have to be determined at run-time are terminated by a 128-bit zero

[5] For several consecutive I/O operations this might lead to minor register initialisation overhead.
[6] In OS0 writing global variables with explicit TAO slice requires `USE_DEF`.

word. The terminating zero word will not be written if the previous word contains a terminating zero byte.