\Orchestrating a brighter world NEC

VEOS high level design

Revision 3.2

NEC

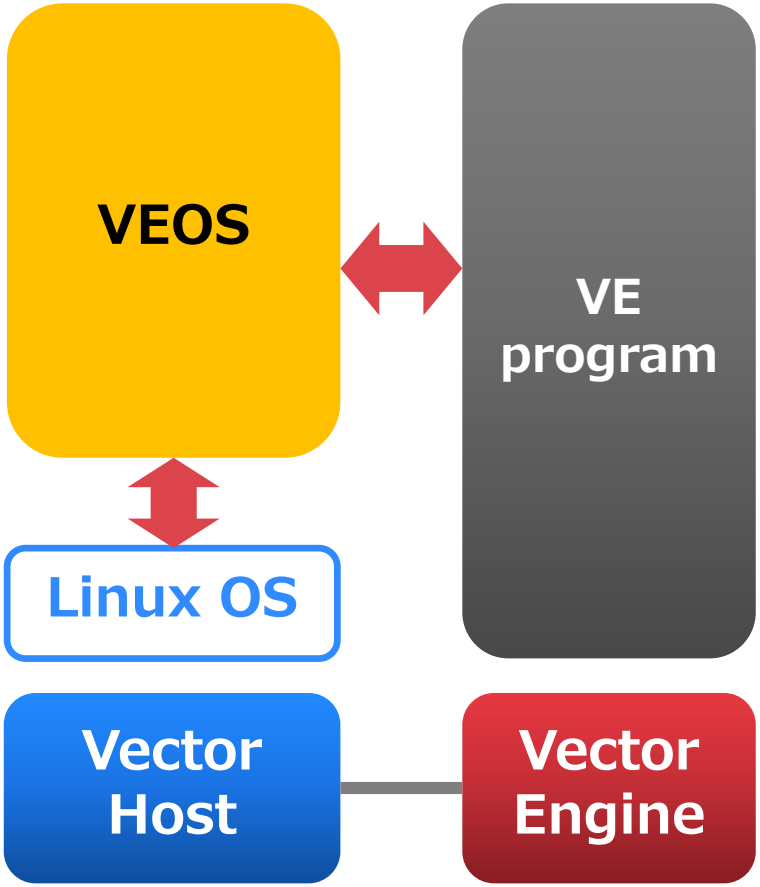
Table of contents

About this document

- This document describes the high level design of VEOS
- This revision covers VEOS v2.2 or later.

What is VEOS

VEOS is the software running on Linux/Vector Host, providing OS functionality for VE programs running on Vector Engine



VEOS features:
- VE program loading
- System call handling
- VE process management
- VE memory management
- Signal handling
- OS commands supporting
  - gdb, ps, free, top, sar etc.

Utilizing Linux capability

Two aspects of VEOS
- From VE prog. : Operating System
- From Linux : Agent for VE program

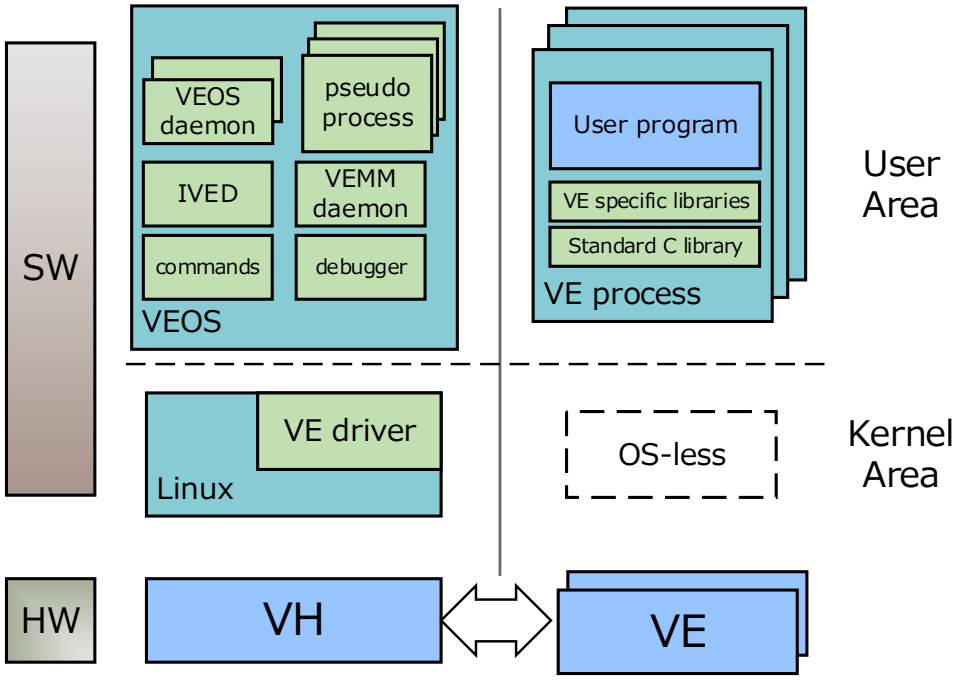
No OS jitter on Vector Engine
- VEOS works completely on Linux/x86.

Basic features of VEOS

Components

VEOS

- VEOS daemon
  - Daemon which manages one VE
  - It is responsible for process management, memory management, etc.
- Pseudo process (ve_exec command)
  - Process which manages one VE process
  - It loads program into VE process and handles system call request from VE process
- IVED
  - Daemon for inter-VE resource management
- VEMM daemon
  - Daemon which communicates with InfiniBand driver
- Commands
  - Ported commands such as “ps” and “free”
- Debugger
  - Ported debugger (gdb)



VE driver

- Driver which provides resource accessibility to VEOS daemon and pseudo process

VE process

- Process which executes user program
- The following libraries may be linked
  - Standard C library
  - VE specific libraries

Process management

Multiprocessing and multithreading are supported

Identification of VE process

- A VE process is identified by PID of the corresponding pseudo process.

Process state

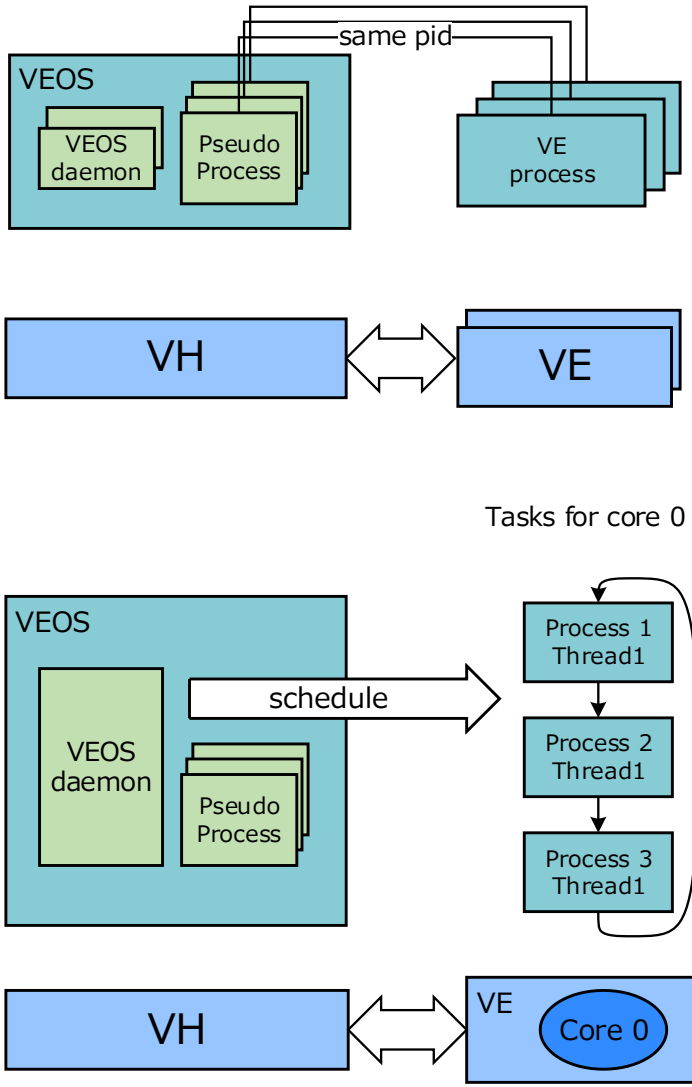
- VEOS maintains the state of VE processes independently from Linux
  - running, wait, etc.

Scheduling

- VEOS chooses a VE core to execute a thread of a VE process, when the thread is started
- VEOS schedules threads of VE processes using round-robin scheduling on each VE core
- Migration of a thread between VE cores may occur when a VE core has no thread to execute

Relationship of parent process and child processes

- VEOS maintains the relationship of VE processes, in order to inherit resource limit, CPU affinity, etc.
  - VEOS maintains the relationship within one VE



Memory management

Virtual address management

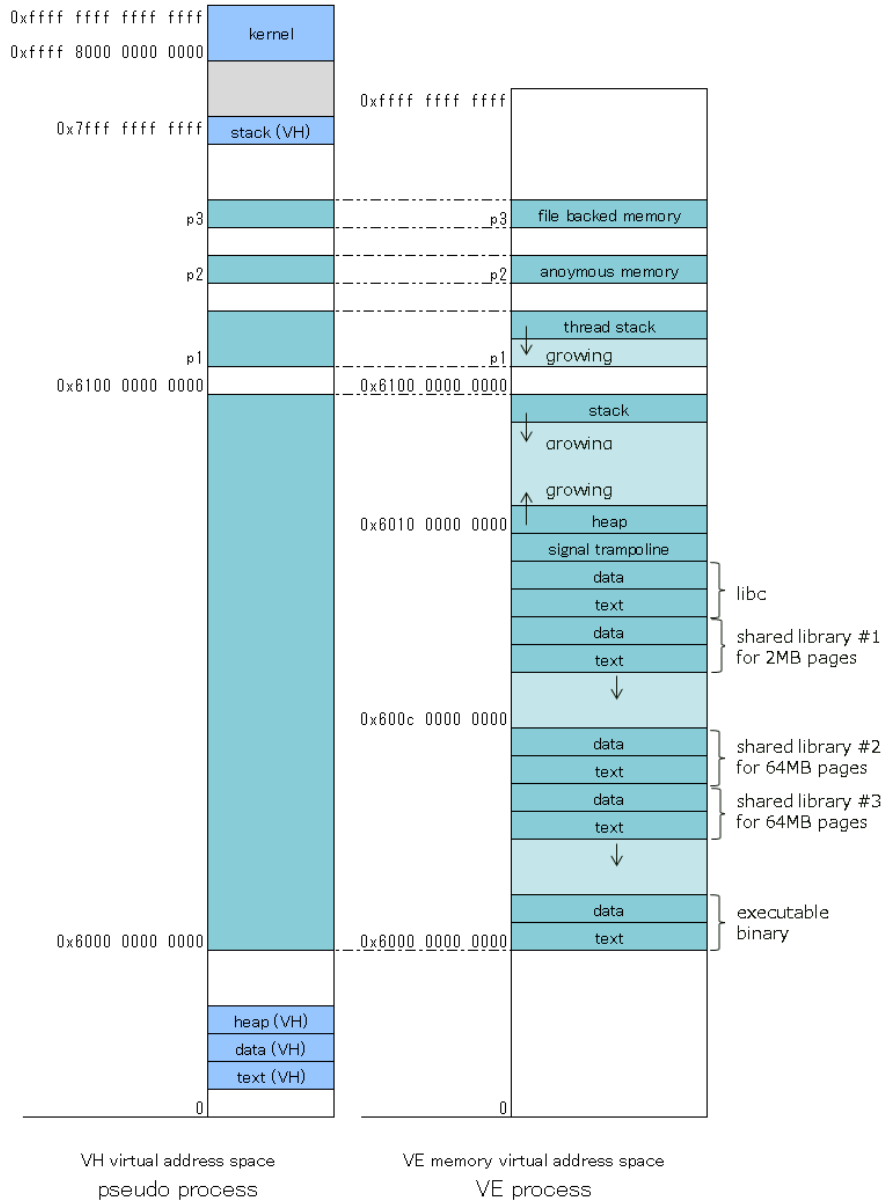
- Text and data of executable binary, heap and stack are located at the fixed address
- Other areas such as anonymous memory are located at address allocated dynamically
- VE memory virtual address space is subset of virtual address space of pseudo process
- Anonymous memory and file backed memory are supported
- Data of file backed memory is transferred between VE and VH by system calls such as mmap(), munmap(), msync() and exit()

Physical VE memory management

- VEOS allocates physical memory when executable binary or shared library are loaded, or anonymous or file backed memory are requested
- Demand paging and copy-on-write are not supported, because HW doesn't support precise exceptions
- Anonymous memory and file backed memory share physical memory between VE processes if shared mapping is requested
- Anonymous memory and file backed memory may share physical memory between VE processes if private mapping with read-only protection is requested
- Text of executable binary and shared library are private file backed memory with read only protection. So, they shares physical memory between VE processes
- Growing heap and stack is supported

Page size

- 64 MB / 2MB
- The page size of text and data of executable, heap, and stack are equal to the alignment of a executable binary
- The page size of text and data of shared library is equal to the alignment of shared library
- The default page size of anonymous memory or file backed memory is equal to the alignment of executable binary. VE program can specify page size when it requests them



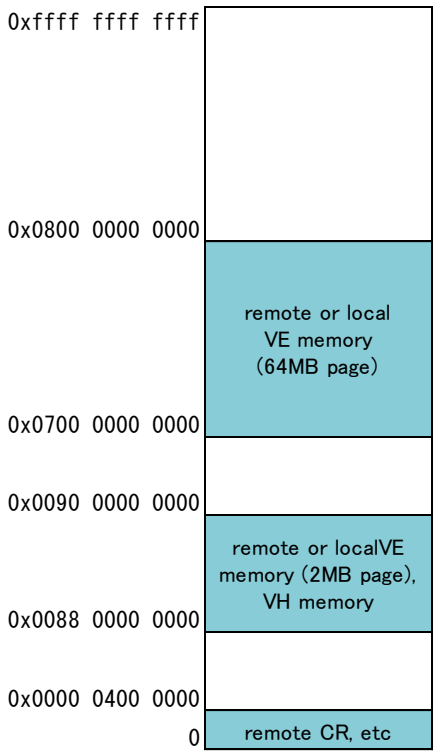
User mode DMA and communication register

User mode DMA

- VE process can use user mode DMA in order to transfer data between memory described below
  1. VE memory assigned to a VE process and VE memory assigned to another VE process
  2. VH memory and VE memory assigned to a VE process
- A VE process can use two DMA descriptor tables
- User mode DMA is performed while one or more threads of the VE process are running on VE cores
- User mode DMA is stopped while all threads of the VE process are not running on VE cores
- Target and source address of DMA is specified using special address named VE host virtual address
- VEOS sets the mapping from VE host virtual address to memory

Communication register (CR)

- CRs are 64-bit registers used for control of exclusive execution or synchronous operation between threads of a process or MPI processes
- VE process can access CR of local VE and remote VE
  - Access to CR of local VE
    - CR access instruction can be used
    - CR is specified using special address named effective CR address
  - Access to CR of remote VE
    - Host memory access instructions can be used
    - CR is specified using VE host virtual address which is also used for user mode DMA
- VEOS sets the mapping from effective CR address or VE host virtual address to actual CR



VE host virtual address space

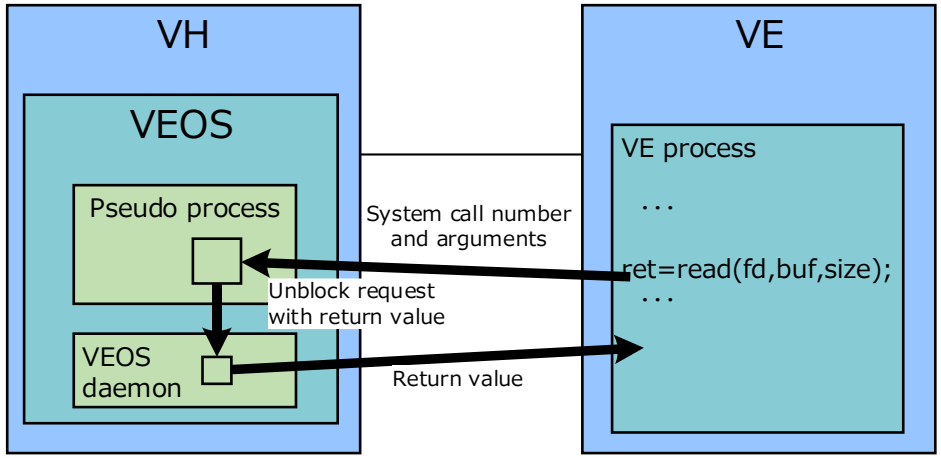
effective CR address space

Notice: User mode DMA and CR are used by the MPI library or other system libraries.
Low level API for user mode DMA is availbale.
CR is not intended to be used directly from user programs.

System call

VE process offloads system call onto VH

- Pseudo process handles request of system call
- Pseudo process requests VEOS daemon or Linux if needed
- System call is handled with pseudo process's privilege



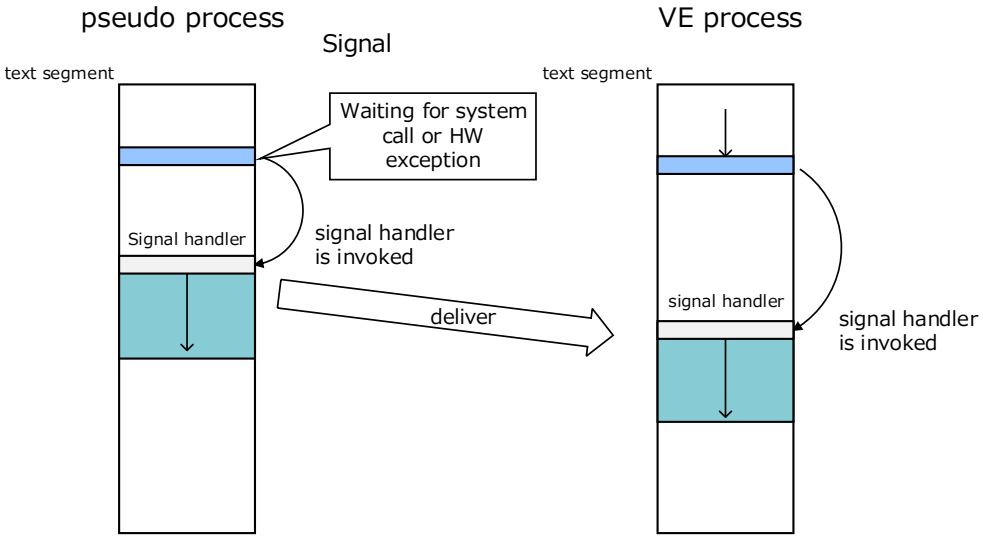
Sequence of system call (simple case)

1. VE process stores system call number and arguments to VH memory
2. VE process stops the VE core and raise an interrupt
3. Pseudo process handles system call
4. Pseudo process sends unblock request with return value to VEOS daemon
5. VEOS daemon stores return value to the register of the VE core and starts the VE core

Signal

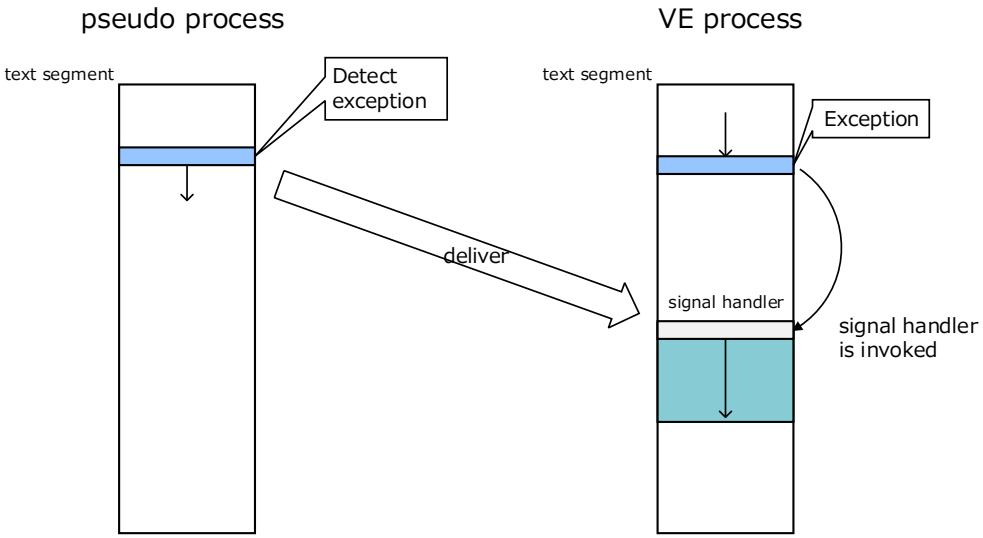
Signal sent by another process

- Signal handler of pseudo process sends a signal request to VEOS
- VEOS delivers signal to VE process when VE process executed on VE core next time
- Pseudo process is killed by SIGKILL if VE process is terminated by signal



Signal due to HW exception

- Pseudo process detects HW exception and sends a signal request to VEOS
- VEOS delivers signal to VE process when VE process executed on VE core next time
- If VE process registers a signal handler corresponding to non-recoverable HW exception, VE process is terminated after execution of the signal handler



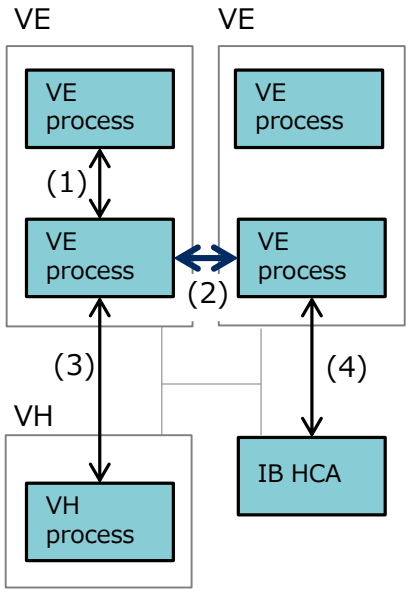
Feature list (Fundamental features)

Category	Feature	Design	Description
Program loader	File format	ELF, DWARF	Program loader recognizes ELF file and loads text and data to VE.
	Dynamic linking	Supported	Dynamic linking of shared library is supported.
	Dynamic loading	Supported	Dynamic loading of shared library is supported. (dlopen(), dlsym(), etc)
	Alignment of segment	2MB / 64MB	Alignment of segment is decided on linking.
Process management	Multiprocessing	Fork-exec model	VE program can create a process using fork() system call. VE program can execute a new VE program using execve() system call.
	Multithreading	POSIX thread	VE program can create threads using POSIX API.
	Scheduling	Round robin on each VE core	Threads are executed in circular order without priority. Threads in blocked state are skipped.
	Preemption	Supported	If the time slice is expired, a context switch occurs, and a next thread starts to run on VE core.
	Time slice	1 second	The period of time for which a thread of a process is allowed to run.
	Timer interval	100 milliseconds	The interval between invocations of the timer handler of the scheduler at VH side.
Memory management	Virtual address space	Supported	Process has own virtual address space.
	Memory allocation	Dynamic	Memory is allocated when VE program is loaded. Growing heap and stack are supported. Allocating anonymous or file-backed memory is also supported.
	File backed memory	Supported	Data of the file is transferred between VE and VH by system calls.
	Demand paging	Not supported	Physical memory is allocated when virtual memory is allocated.
	Copy on write	Not supported	Private readable and writable area has own physical memory.
	Sharing physical memory	Supported	Anonymous memory and file backed memory share physical memory under certain conditions.
	Page size	2MB / 64MB	Page size of segments of executable binary and page size of shared library is equal to the alignment of them. VE program can specify page size of anonymous or file backed memory.
Input and output	File system, network, etc	Supported	Offloaded to Linux.

Feature list (inter process communication)

Feature	(1) Inside VE	(2) VE-VE	(3) VE-VH	(4) VE-IB	Function name example
System V shared memory	✓	-	-	-	shmget
POSIX shared memory	✓	-	-	-	shm_open
Spin lock	✓	-	-	-	pthread_spin_init
Mutex	✓(*1)	-	-	-	pthread_mutex_init
Read write lock	✓	-	-	-	pthread_rwlock_init
Condition variable	✓	-	-	-	pthread_cond_init
Barrier	✓	-	-	-	pthread_barrier_init
Unnamed semaphore	✓	-	-	-	sem_init
System V semaphore	✓	✓	✓	-	semget
Named semaphore	✓	✓	✓	-	sem_open
System V message queue	✓	✓	✓	-	msgget
POSIX message queue	✓	✓	✓	-	mq_open
UNIX socket	✓	✓	✓	-	socket
Pipe, Fifo	✓	✓	✓	-	pipe, mkfifo
Signal	✓	✓	✓	-	kill
VH-VE SHM *2	-	-	✓	-	vh_shmat
VESHM *3 *5	✓	✓	-	-	-
CR *5	✓	✓	-	-	-
VEMM *4 *5	-	-	-	✓	-

✓ Supported - Not supported



IB HCA: InfiniBand HCA

*1 robust mutex and pi mutex are not supported
*2 VH-VE SHM is a feature to allow VE processes to transfer data to or from System V shared memory created at VH side
*3 VESHM is a feature to allow VE processes to transfer data between VE processes
*4 VEMM is a feature to allow the IB driver to access VE memory and to allow VE programs to access IB HCA
*5 These features are used by the MPI library or other system libraries. They are not intended to be used directly from user programs

Available resources

Item	Condition	Per VE	Per VE process	Note
Maximum number of VE process	-	256 processes	-	-
Maximum number of threads	-	1,024 threads	64 threads	
Maximum number of connections VEOS daemon accepts	-	1,056	-	A connection between a VEOS daemon and a pseudo process is established per a VE thread. A connection between a VEOS and a ported command / gdb and is established per a request.
Maximum size of memory assigned to VE process	48GB memory model	49,024MB *2	49,024MB *2	A 64MB-page is reserved for zeroed page. A 64MB–page is reserved for synchronization. *4
	24GB memory model	24,448MB *3	24,448MB *3	
Maximum size of accessible remote VE memory	8 cores model	894GB	894GB	The value will be changed when the number of cores is changed.
Descriptor table of user mode DMA	-	-	2 descriptor tables	-
Maximum number of CR page for threads *1	-	-	1 page	-
Maximum number of local CR pages for MPI *1	8 cores model	24 pages	3 pages	The value will be changed when the number of cores is changed.
Maximum number of remote CR pages for MPI *1	-	-	32 pages	-

*1 A CR page consists of 32 CRs
*2 49,086MB when VE is connected to SX-Aurora TSUBASA A100-1 model
*3 24,510MB when VE is connected to SX-Aurora TSUBASA A100-1 model
*4 A 2MB-page is reserved for synchronization when VE is connected to SX-Aurora TSUBASA A100-1 model

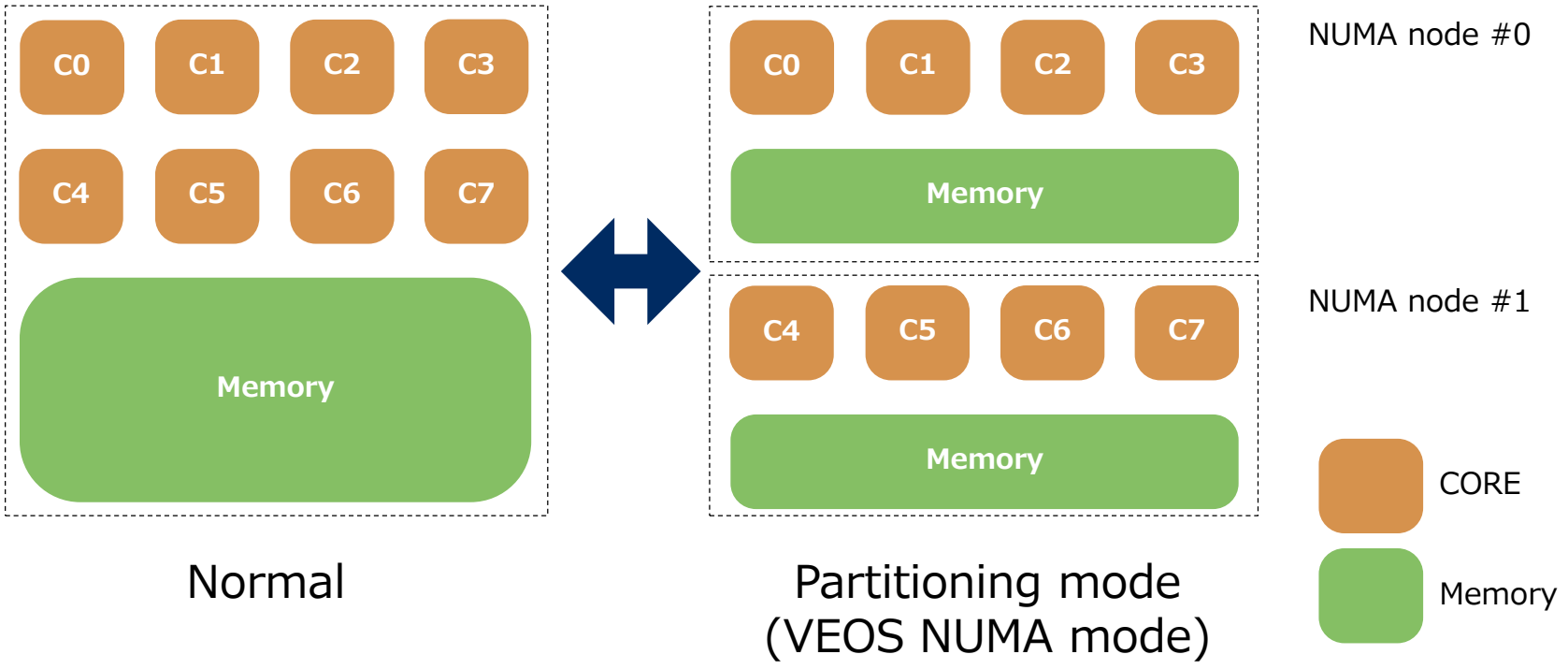
Notice

1. The number of threads of VE processes which run on VE should be less than or equal to the number of available VE cores, in order to achieve best performance
2. VE process can execute new VE program or VH program. When VE process executes new VE program, VE process needs to specify VE program with the first argument of execve() system call. When VE process executes VH program, VE process's information such as resource limit is discarded, even if VH program executes VE program, again
3. Almost all of system calls, standard C library functions, Linux's commands and debugger commands are supported. But, some of them are not supported

Extended features

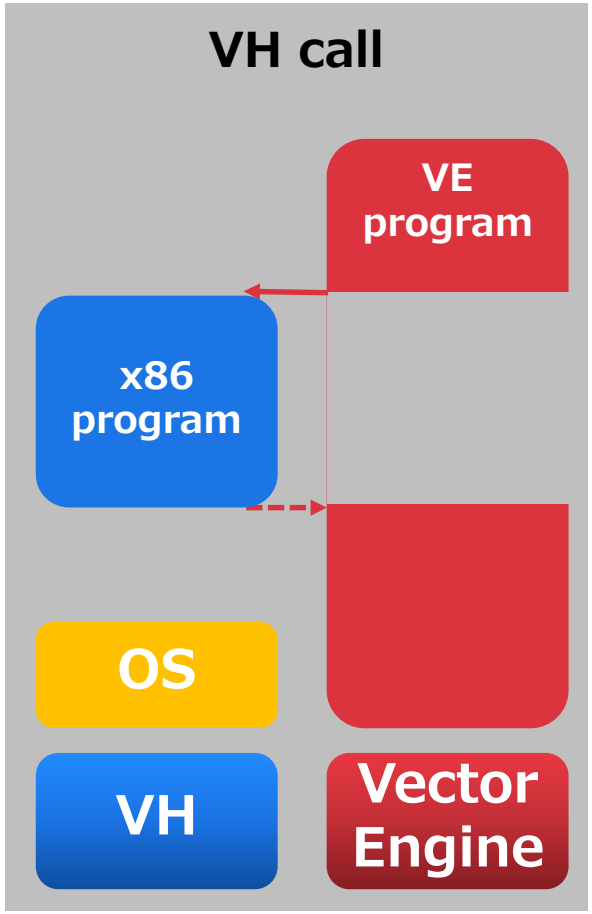
Support of VE partitioning mode (VEOS NUMA mode)

- VE supports the partitioning mode in which the VE cores, last level caches and its main memory are partitioned into two segments
- VEOS treats two segments in partitioning mode as two NUMA nodes in order to utilize existing system call interfaces and commands
- VEOS assigns cores and memory to VE processes considering the locality



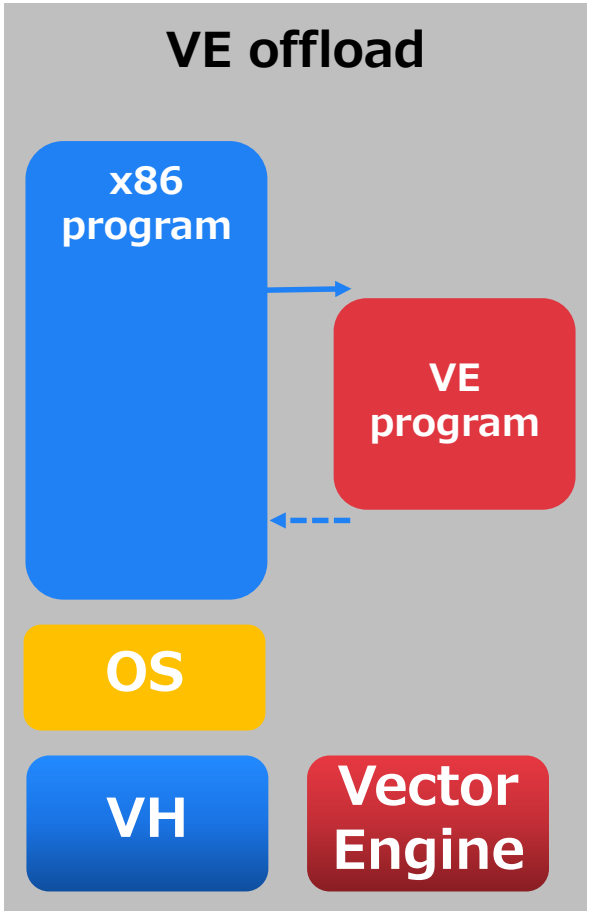
Support of offloading programing model

- “VH call” offloads scalar operations from VE to x86



VH call is synchronous API

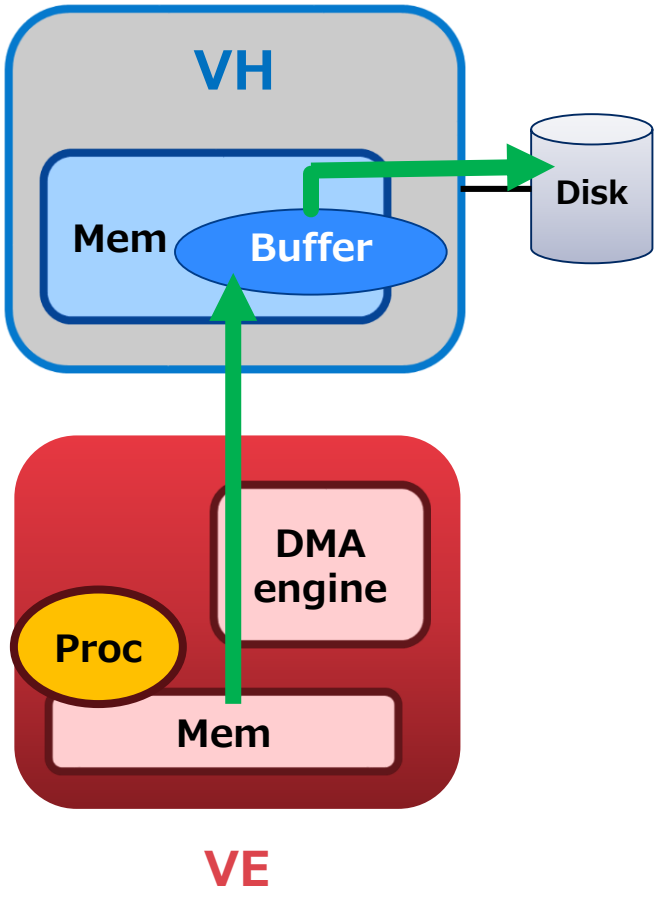
- “VE offload” offloads compute kernels from x86 to VE



VE offload is asynchronous API

Accelerated I/O

- The throughput and the latency of read/write family are improved
  - read, pread, readv, preadv
  - write, pwrite, writev, pwritev
- Accelerated I/O transfer data between VE and VH by VE DMA engine
  - VE DMA engine directly accesses the buffer at VH side.
- To use Accelerated I/O, the following environment variable setting is needed

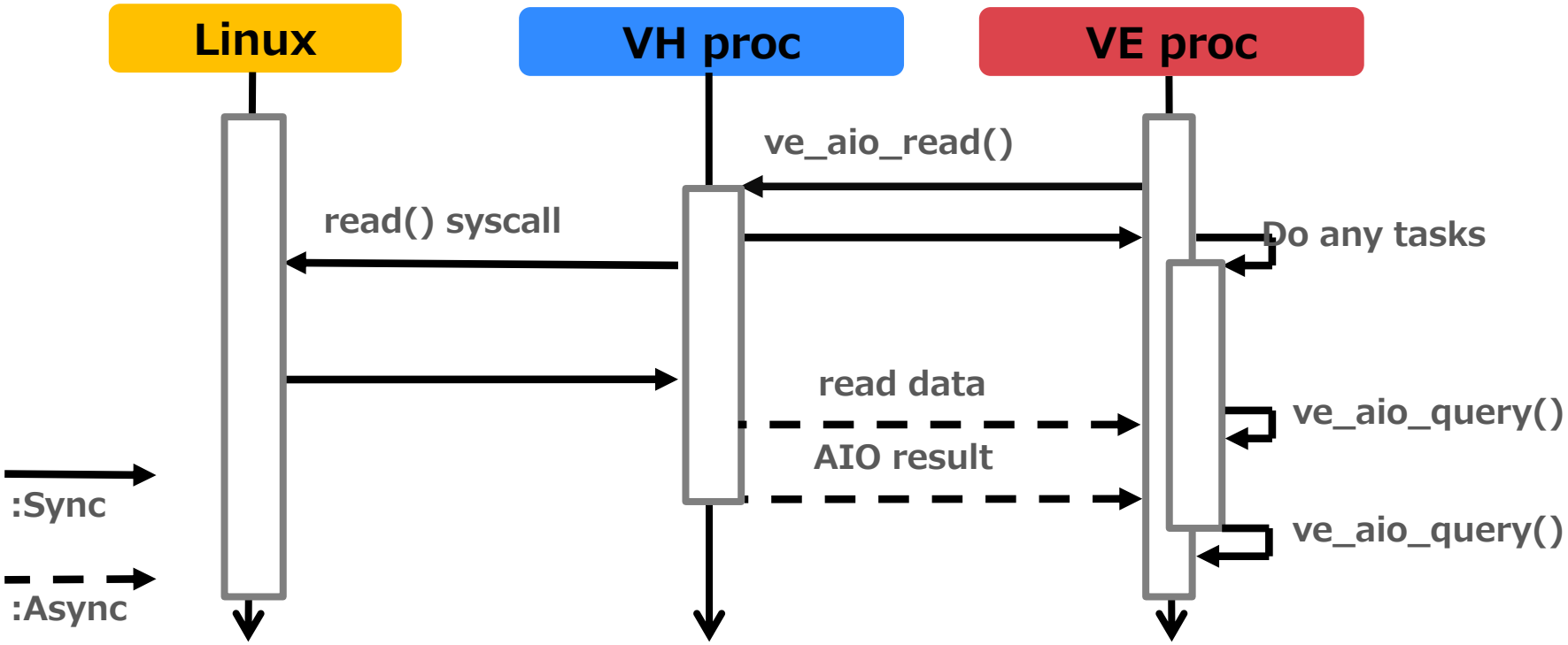



```
(In case of bash)
export VE_LD_PRELOAD=libveaccio.so.1
```

Asynchronous I/O

- Asynchronous I/O for VE (VE AIO) enables VE programs to do their tasks during I/O system call operation

- VE AIO doesn't need any VE resource while the I/O operation on VH is in progress.



Revision History

Revision History

Rev.	Date	Update
1.8	28th Feb. 2018	The first version
1.9	14th May 2018	• Updated the introduction of VEOS to clarify that it is running on Linux/VH (Page 4) • Added VH-VE SHM which is a new feature of VEOS 1.1 release (Page 12)
1.10	11st July 2018	• Mentioned migration of a thread between VE cores occurs after VEOS 1.2.1 (Page 6) • Updated the description of maximum size of memory (Page 13) • Incorporated the change of the number of DMA descriptor table done by VEOS 1.2 (Page 13)
1.11	12nd Sep. 2018	• Updated the description of user mode DMA because Low level API for user mode DMA is provided as an experimental feature (Page 8) • Updated the feature list because VE programs can use VH-VE SHM (Page 12) • Added fifo to the feature list (Page 12) • Improved figures
2.0	5th Dec. 2018	• Removed notice relating to kernel headers because kernel headers are provided as a part of VEOS 2.0 or later (Page 14)
2.1	9th Feb. 2019	• This revision covers VEOS v2.0.1 or later. • Changed the format of top page. • Added information which shows target VEOS versions of this document. (Page 3)
3.0	22nd Apr. 2019	• This revision covers VEOS v2.1 or later. • Updated “What is VEOS” (Page 4) • Mentioned Pseudo process is ve_exec command (Page 6) • Low level API for user mode DMA is not experimental feature currently (Page 9) • Described extended features (Page 17-20)
3.1	19th June 2019	• Added descriptions of VEOS NUMA mode (Page 17)
3.2	Sep. 2019	• This revision covers VEOS v2.2 or later. • Updated memory address map (Page 8)