システムコールの相違点

Revision 3.1

更新履歴

Rev.	発行日	更新内容
2.2	18/2/28	初版
2.3	18/3/23	6 章 シグナル関連の相違点について更新
2.4	18/7/23	2.9 節 ファイルと関連付けられた mmap() に関する相違点-15 を追加 6 章 VE 実行ファイルの set-user-id、 set-group-id ビットの挙動について追加
2.5	18/8/15	2.18 節 sigprocmasc() に関する相違点を追加 2.19 節 clock_gettime() に関する相違点を追加 6.19 節 /proc/self に関する相違点を追加
2.6	18/9/14	6.23 節 write() と類似システムコールに関する相違点を追加
2.7	18/12/13	2.12 節 execve() システムコールの argv/envp 数について更新 glibc に関する相違点を追加
2.8	19/2/8	この版は VEOS v2.0.3 に対応します 表紙の書式を変更 6.24 節 非アトミック I/O について追加 6.25 節 prlimit() システムコールと RPM コマンドの動作について追加
2.9	19/4/15	この版は VEOS v2.1 以降に対応します 2.12 節 execve() システムコール実行時、常に設定される環境変数について更新 Musl-libc に関連する相違点を削除
3.0	2020/5	この版は VEOS v2.5 以降に対応します 6 章 RSS の制限の相違点について更新
3.1	2020/7	この版は VEOS v2.6.2 以降に対応します 2.11 節 execve() システムコールの argv/envp 数について更新(日本語版のみ) 2.19 節 times()システムコールについて追加 2.20 節 acct()システムコールについて追加 6.25 節 シグナルに関連する相違点を追加 6 章 コマンドライン引数と環境変数の数の最大値について追記

1. 導入

このドキュメントは、Linux のシステムコールと VEOS システムコールの実行に関するすべての相違点の記述を目的として作成されています。

すべてのシステムコールは下記のとおり分類されています:

1. サポートされているシステムコール
   ここでは VEOS で完全にサポートされているシステムコールの一覧と、Linux のシステムコールに対する相違点が記載されています。

2. 部分的にサポートされているシステムコール
   ここでは VEOS で部分的にサポート(制限付)されているシステムコール一覧と、Linux のシステムコールに対する相違点が記載されています。

3. サポートされていないシステムコール
   ここでは VEOS でサポートされていないシステムコールの一覧が記載されています。

2. サポートされているシステムコール

以下は VEOS で完全にサポートされているシステムコールの一覧です。

SL No	システムコール	相違点（あり/なし）
1.	fork	あり
2.	waitid	あり
3.	sched_getaffinity	あり
4.	sched_setaffinity	あり
5.	sched_yield	なし
6.	getpgrp	なし
7.	getpid	なし
8.	getpgid	なし
9.	getppid	なし
10.	gettid	なし
11.	getsid	なし
12.	setsid	なし
13.	setpgid	なし
14.	time	なし
15.	gettimeofday	なし
16.	clock_getres	なし
17.	vfork	なし
18.	exit	なし
19.	execve	あり
20.	sysinfo	なし
21.	sched_rr_get_interval	あり
22.	acct	なし
23.	clock_gettime	あり
24.	kill	なし
25.	tkill	なし
26.	tgkill	なし
27.	rt_sigqueueinfo	なし
28.	rt_tgsigqueueinfo	なし
29.	sigaction	なし
30.	sigprocmask	あり
31.	sigreturn	なし
32.	sigsuspend	なし
33.	sigaltstack	あり
34.	sigpending	なし

35.	signalfd	あり
36.	signalfd4	あり
37.	rt_sigtimedwait	なし
38.	lookup_dcookie	なし
39.	semtimedop	なし
40.	recvmmsg	あり
41.	timer_getoverrun	なし
42.	sendmsg	あり
43.	name_to_handle_at	なし
44.	mq_getsetattr	なし
45.	open_by_handle_at	なし
46.	inotify_add_watch	なし
47.	timerfd_settime	なし
48.	timerfd_gettime	なし
49.	newfstatat	なし
50.	inotify_rm_watch	なし
51.	ioprio_set	なし
52.	ioprio_get	なし
53.	ppoll	なし
54.	getsockopt	なし
55.	poll	なし
56.	epoll_ctl	なし
57.	getgroups	なし
58.	socketpair	なし
59.	fanotify_mark	なし
60.	readlink	なし
61.	epoll_create1	なし
62.	fanotify_init	なし
63.	semctl	なし
64.	recvmsg	なし
65.	writev	なし
66.	msgctl	なし
67.	msgrcv	なし
68.	recvfrom	あり
69.	mount	なし
70.	truncate	なし
71.	getpeername	なし
72.	mq_timedreceive	あり

73.	accept4	なし
74.	sendto	なし
75.	accept	なし
76.	mq_timedsend	あり
77.	utimensat	なし
78.	epoll_pwait	なし
79.	splice	なし
80.	getresgid	なし
81.	utime	なし
82.	mq_open	なし
83.	symlink	なし
84.	statfs	なし
85.	renameat	なし
86.	epoll_wait	なし
87.	utimes	なし
88.	symlinkat	なし
89.	flock	なし
90.	futimesat	なし
91.	connect	なし
92.	msgsnd	なし
93.	readlinkat	なし
94.	setdomainname	なし
95.	getdents	なし
96.	mq_notify	なし
97.	uname	なし
98.	setsockopt	なし
99.	fcntl	なし
100.	setgroups	なし
101.	syslog	なし
102.	access	なし
103.	openat	なし
104.	write	なし
105.	pwritev	なし
106.	pwrite64	なし
107.	sethostname	なし
108.	creat	なし
109.	fstatfs	なし
110.	open	なし

111.	stat	なし
112.	bind	なし
113.	setuid	なし
114.	fstat	なし
115.	getcwd	なし
116.	timer_gettime	なし
117.	setgid	なし
118.	ftruncate	なし
119.	close	なし
120.	pause	なし
121.	socket	なし
122.	eventfd2	なし
123.	fdatasync	なし
124.	vhangup	なし
125.	fadvise64	なし
126.	inotify_init	なし
127.	epoll_create	なし
128.	select	なし
129.	unlink	なし
130.	pselect6	なし
131.	dup	なし
132.	dup2	なし
133.	pipe	なし
134.	nanosleep	なし
135.	chown	なし
136.	lchown	なし
137.	fchown	なし
138.	lseek	なし
139.	mkdir	なし
140.	tee	なし
141.	chroot	なし
142.	ioperm	なし
143.	alarm	なし
144.	mknodat	なし
145.	setreuid	なし
146.	sync	なし
147.	getgid	なし
148.	sync_file_range	なし

149.	mknod	なし
150.	fsync	なし
151.	rename	なし
152.	dup3	なし
153.	faccessat	なし
154.	lstat	なし
155.	readahead	なし
156.	getsockname	なし
157.	preadv	なし
158.	pread64	なし
159.	read	なし
160.	mq_unlink	なし
161.	semget	なし
162.	linkat	なし
163.	setresuid	なし
164.	eventfd	なし
165.	fchmodat	なし
166.	umask	なし
167.	fchmod	なし
168.	fchownat	なし
169.	readv	なし
170.	link	なし
171.	rmdir	なし
172.	setfsgid	なし
173.	setfsuid	なし
174.	chmod	なし
175.	chdir	なし
176.	geteuid	なし
177.	pipe2	なし
178.	unlinkat	なし
179.	setregid	なし
180.	msgget	なし
181.	listen	なし
182.	fchdir	なし
183.	semop	なし
184.	getresuid	なし
185.	inotify_init1	なし
186.	iopl	なし

187.	fallocate	なし
188.	getegid	なし
189.	mkdirat	なし
190.	setresgid	なし
191.	getuid	なし
192.	getdents64	なし
193.	timerfd_create	なし
194.	umount2	なし
195.	timer_delete	なし
196.	shutdown	なし
197.	syncfs	なし
198.	pivot_root	なし
199.	mmap	あり
200.	munmap	なし
201.	mprotect	なし
202.	msync	あり
203.	shmget	あり
204.	shmat	なし
205.	shmctl	あり
206.	process_vm_readv	なし
207.	process_vm_writev	なし
208.	grow	あり
209.	getrusage	あり
210.	sendfile	なし
211.	timer_settime	なし
212.	Sendmmsg	あり
213.	brk	なし
214.	shmdt	なし
215.	fgetxattr	あり
216.	flistxattr	あり
217.	fremovexattr	あり
218.	fsetxattr	あり
219.	getxattr	あり
220.	lgetxattr	あり
221.	listxattr	あり
222.	llistxattr	あり
223.	lremovexattr	あり
224.	lsetxattr	あり
225.	removexattr	あり

226.	setxattr	あり
227.	sysve	VEOS 特有
228.	times	あり

1. waitid()

1. 子に終了シグナル(例：SIGFPE, SIGTERM など)が送られる場合、実際のシグナルコールの代わりに SIGKILL が送られます。これにより、子プロセスが親で待機している際に、WTERMSIG（status）value = SIGKILL（9）となります。

2. fork()

1. VEOS ではコピーオンライトはサポートされていません。子プロセスが作成されたとき新規メモリが割り当てられます。
2. プロセスがオープンファイル記述子の最大限度を使い切ると、fork()システムコールのその後の呼び出しは失敗し、errno は EAGAIN に設定されます。
3. chroot()システムコールを使用して呼び出したプロセスのデフォルトルートディレクトリを path に指定したディレクトリに変更すると、その後の fork()の呼び出しは失敗し、errno は EAGAIN に設定されます。

3. sched_getaffinity()

1. pid1 が引数として与えられる場合、-1 がリターンされ errno ESRCH が設定されます。

4. sched_rr_get_interval()

pid1 が引数として与えられる場合、-1 がリターンされ errno ESRCH が設定されます。

5. sigaltstack()

1. 代替スタックの最小サイズは VE_MINSIGSTKSZ (533400)となります。ユーザが 512KB 以下のスタックサイズを設定した場合 ENOMEM がリターンされます。
2. 無効なスタックポインタでスタックの登録が試みられた場合、sigaltstack()は EFAULT で失敗します。

6. sched_setaffinity()

1. pid1 が引数として与えられる場合、-1 がリターンされ errno ESRCH が設定されます。

7. Signalfd()/signalfd4

無効な「マスク」引数で signalfd()が呼び起こされる場合、EINVAL の代わりに EFAULT がリターンされます。

8. mmap()

1. VE では以下の mmap()のフラグはサポートされておらず、EINVAL がリターンされます。
    a. MAP_GROWSDOWN
    b. MAP_HUGETLB
    c. MAP_LOCKED
    d. MAP_NONBLOCK
    e. MAP_POPULATE
2. VEOS では、hugetlb ファイルシステムを介した huge page マッピングは実装されていません。
3. mmap では二種類のページサイズのみサポートされており、最小のページサイズは 2MB、最大のページサイズは 64MB となります。

4. 新しいフラグには、MAP_2MB、MAP_64MB が追加され、特定のページサイズでメモリ・マッピングが行われます。
5. VE プロセスが mmap フラグで特定のページサイズを指定しない場合、実行可能なページサイズに基づきデフォルトのページサイズが決定されます。
6. MAP_STACK はサポートされており、grow()システムコールと共に使用されます。それ以外の動作は実装されていません。このフラグでは物理マッピングは行われません。
7. ユーザが 96TB-97TB の範囲で MAP_FIXED フラグと mmap()の実行を試みた場合、この範囲は VE プロセスアドレス空間に対して既に確保されているため、失敗します。
8. MAP_SHARED フラグ付きのファイルと関連付けられた mmap()の場合、VE メモリは同じ VE ノードの VE プロセスによって共用されます。指定されたファイル内容を含む VE メモリの内容は、msync()、munmap()システムコールの実行、または VE プロセスの終了時に同期されます。このアーキテクチャにより、VE プロセスが msync()または munmap()を呼び出すか、終了するまで、その VE メモリ内容の変更は別の VE ノード上のプロセスまたは VH プロセスからは見られません。
9. MAP_SHARED フラグ付きファイルと関連付けられた mmap()の場合、mmap 要求が最初に来ると、ファイルの内容が VE メモリに転送されます。すでにマップされている領域の mmap 要求が再び発生した場合、ファイルの内容は VE メモリに同期されません。 したがって、別の VE、あるいは VH プロセスの VE プロセスによる下線ファイルの変更は、VE プロセスでは見られません。
10. MAP_SHARED フラグ付きのファイルと関連付けられた mmap()の場合、VE プロセスがファイルをマップすると（これをマッピング 1 とする）、マッピング 1 にアクセスしているときにファイルサイズを小さくするために ftruncate()を呼び出しても SIGBUS は生成されません。しかし同じファイルへの新しい mmap()において、その変更後のファイルサイズを超えてアクセスしようとした場合には、SIGBUS が生成されます。
11. MAP_NORESERVE フラグがユーザによって指定されたとしても、VE 物理ページは、利用可能な VE メモリに基づいて入力されたサイズによって割り当てられます。
12. ファイルが異なったページサイズでマップされている場合、VE メモリは共有されません。
13. VEOS は、shm_open()によって作成された POSIX 共有メモリオブジェクトを、ファイルバックメモリと同じ方法で処理します。 したがって、1 つの VE ノードにおいて共有メモリとして使用することができます。 この時、コンテンツを保存するためのスワップ領域が必要です。
14. VE では、ファイル "/ dev / zero"でマップされたメモリへのアクセスにより SIGBUS が与えられますが、Linux では成功となります。 そのため、サイズがゼロであるすべてのタイプのファイルにおいて、VE でマッピングされアクセスされる際に常に SIGBUS が発生します。
15. ファイルと関連付けられた mmap()において、ファイルへの書き込みが行われたとしても、VE メモリへの同期は行われません。したがって、VE プロセスが write()システムコールによって関連付けられているファイルを変更したとしても、VE プロセスからはその変更が見えません。

9. shmget()

1. SHM_HUGETLB フラグは VEOS ではサポートされておらず、VE プロセスがシステムコールでこのフラグを使用した場合 EINVAL がリターンされます。
2. shmget は二つのページサイズのみサポートされており、shm ページサイズの最小値は 2MB で最大値は 64MB となります。

3. 新しいフラグに SHM_2MB、SHM_64MB が追加され、指定されたページサイズの共有メモリセグメントが作成されます。
4. ユーザにより SHM_NORESERVE フラグが指定されたとしても、VE 物理ページは利用可能な VE メモリに基づいたセグメントサイズにより割り当てられます。
5. 共有メモリセグメントのサイズアラインメントの最小サイズは SHMLBA (4KB)と同じではありません。サイズアラインメントは SHM_2MB/SHM_64MB によります。
6. VE メモリは、同じ VE ノードの VE プロセスによって共有されます。 VE メモリと VH メモリは同期されません。 したがって、VE メモリのコンテンツは、別の VE ノードまたは VH プロセスの VE プロセスからは見られません。
7. プロセス 1 が 64MB のページサイズの共有メモリを作成し、他のプロセス 2 が 2MB のページ・サイズの shmget()を呼び出す場合、セグメントがすでに作成されているため shmget()は成功します。しかし、このセグメントをプロセス 2 に添付している間に返されるアドレスは 64MB にアラインされます。（作成中に定義されたセグメントページサイズによる。）

10.grow()

grow()システムコールは、VE プロセスまたはスレッドのスタックサイズの拡大のために使用されます。

1. システムコールには引数が二つあります。
2. 無効なアドレスが指定された場合、システムコールは EINVAL をリターンします。
3. このシステムコールは、関数 epilogue / prologue を介して呼び出されます。 VE プロセスが明示的にこのシステムコールを呼び出すことは推奨されていません。（呼び出す際の動作は定義されていません。）

11.execve()

```
int execve(const char *filename, char *const argv[], char *const envp[]);
```

1. execve()システムコールが成功すると、新しくロードされた VE バイナリの argv [0]には常に VE バイナリの絶対パスが格納されます。execve()の呼び出し側が argv [0]に何らかのデータを格納すると、その内容は失われます。したがって、execve()の呼び出し側は、argv [0]にバイナリ名を渡すという規則に厳密に従わなければなりません。なお、その他引数（argv [1]、argv[2]、・・・）は VE プログラムへ渡されます。

   注意:これはすべての exec()関数群に適用されます。

2. VE プロセスは新規 VE プログラムや VH プログラムを実行することができます。VE プロセスが新規 VE プログラムを実行する際、VE プロセスは execve()システムコールの最初の引数で VE プログラムを指定する必要があります。VE プロセスが VH プログラムを実行するとき、VH プログラムが再度 VE プログラムを実行しても、VE プロセスのリソースの制限などの情報は破棄されます 。

3. execve()システムコールの二つ目の引数“**argv**”は新規プログラムへ引き渡される引数文字列の配列です。VE において引渡し可能な引数文字列の数は、最大で 512 です。もし、512 個を超える引数文字列が渡された場合は、errno に E2BIG がセットされてシステムコールが失敗します。

   execve()システムコールの三つ目の引数”**envp**”はキー=値の形式をもつ文字列の配列です。環境変数の数は、最大で512です。もし、512個を超える環境変数が渡された場合は、errnoにE2BIGがセットされてシステムコールが失敗します。この環境変数の個数は、常に新しいプログラムに

引き継がれる 6 つの環境変数(VE_EXEC_PATH、 LOG4C_RCPATH、 HOME、 PWD、 VE_LD_ORIGIN_PATH、 VE_NODE_NUMBER)を含んでいます。

4. execve()システムコールに対して実行可能なパーミッションを持つ無効なファイルを渡すと、スレッドグループ全体が終了します。

12.msync()

1. VEフラグでは、MS_INVALIDATEはサポートされておらず、EINVALとしてエラーが返されます。 同じファイルの他のマッピングを無効にするサポートはありません。

13.recvfrom() / recvmmsg() / sendmmsg() / mq_timedreceive() / mq_timedsend()

VE 環境では、バッファサイズの引数としてサイズ/長さ/カウント size_t (unsigned int)を用いるシステムコールの動作が異なる可能性があります。例えば上記 recvfrom システムコール関連において、システムコールの入力引数として size_t len を使用しています。

```
ssize_t recvfrom(int sockfd, void *buf, size_t len, int flags,
                         struct sockaddr *src_addr, socklen_t *addrlen);
```

recvfrom（2）を呼び出す時、ユーザアプリケーションが len 引数として負の値 "-1"（この値は巨大な正の値に変換されます）を渡すとします。
このシナリオ(マイナスの値が巨大な正の値となる場合)では、システムコールを取り扱う際に Linux カーネルは巨大な正の値を MAX_RW_COUNT へ切り捨て、システムコールは成功となります。
しかしシステムコールの VEOS 実装では、この巨大な正の値を MAX_RW_COUNT に切り捨てた後でも、無効な負の値がシステムコールの引数として渡される（巨大な正の値とみなされる）ため、VEOS 実装のシステムコールハンドラが、ローカルバッファを割り当て、len（MAX_ROW_COUNT になる）の値に基づいて VE メモリからデータを送受信する必要があります。そのためローカルバッファ（malloc 経由）の割り当てが失敗したり、VE メモリからのデータ送受信が失敗する可能性があります。またシステムコールが EFAULT / ENOMEM をエラーコードとして返すことがあります。

同様のシステムコールの動作の違いは、以下のシステムコールでも見られます。

- recvmmsg()/sendmmsg()
- mq_timedreceive()/mq_timedsend()
- lookup_dcookie()
- getsockopt()/setsockopt()
- readv()/writev()
- sendto()
- epoll_pwait()
- epoll_wait()
- setgroups()
- read()/write()
- getcwd()
- pread64()/pwrite64()
- getdents64()

14.shmclt()

1. shmctl()の以下のフラグは VE ではサポートされておらず、EINVAL をリターンします。
    a. SHM_LOCK
    b. SHM_UNLOCK

15.getrusage()

次の利用構造のフィールドは VE では維持されておらず、0 をリターンします。

- struct timeval ru_stime;
- long ru_minflt;
- long ru_majflt;

16.chroot()

注意

❖ 呼び出し元プロセスのデフォルトルートディレクトリを chroot()システムコールで path に指定したものに変更した後、fork()、vfork()、clone()システムコールの後続の呼び出しは errno に EAGAIN をセットして失敗します。

17.sigprocmask()

VEOS において、SIGCONT シグナルはマスクされません。sigprocmask(2) による SIGCONT シグナルマスクの要求は無視され、sigprocmask(2) はユーザプログラムに成功をリターンします。

18.clock_gettime()

引き数 “clock_id” が INIT プロセスの CPU 時刻のクロック ID の時、VEOS は INIT プロセスを持たないため、errno に EINVAL をセットして失敗します。

19.times()

次の times()構造体のフィールドは VE では維持されておらず、0 をリターンします。

- tms_stime
- tms_cstime

20.acct()

これは ve_acct 構造体の定義です。ve_acct 構造体は VE プロセスアカウンティングが有効な場合に VE プロセスアカウンティングファイルに書き込まれるプロセスアカウンティング情報の構造体です。

```
struct ve_acct {
        char            ac_flag;      /* Flags */
        char            ac_version;   /* Always set to ACCT_VERSION */
        unsigned short int ac_tty;      /* Control Terminal */
        unsigned int    ac_exitcode;  /* Exitcode */
        unsigned int    ac_uid;       /* Real User ID */
        unsigned int    ac_gid;       /* Real Group ID */
        unsigned int    ac_pid;       /* Process ID */
```

```
        unsigned int      ac_ppid;      /* Parent Process ID */
        unsigned int      ac_btime;     /* Process Creation Time */
        float             ac_etime;     /* Elapsed Time */
        comp_t            ac_utime;     /* User Time */
        comp_t            ac_stime;     /* System Time. It is always 0 for VE processes */
        comp_t            ac_mem;       /* Memory Usage on termination [kb] */
        comp_t            ac_io;        /* Chars Transferred */
        comp_t            ac_rw;        /* Blocks Read or Written */
        comp_t            ac_minflt;    /* Minor Pagefaults */
        comp_t            ac_majflt;    /* Major Pagefaults */
        comp_t            ac_swaps;     /* Number of Swaps */
        char              ac_comm[16];  /* Command Name */
        unsigned int      ac_sid;       /* session ID */
        unsigned int      ac_timeslice; /* timeslice [us] */
        unsigned short int ac_max_nthread; /* max number of threads */
        unsigned short int ac_numanode;  /* NUMA node number */
        double            ac_total_mem; /* VE's total memory usage in clicks */
        unsigned long long ac_maxmem;    /* VE's max memory usage [kb] */
        unsigned long long ac_syscall;   /* the number of system calls */
        double            ac_transdata; /* data transfer amount between VE-VH [kb] */
        unsigned long long ac_ex;        /* Execution count */
        unsigned long long ac_vx;        /* Vector execution count */
        unsigned long long ac_fpec;      /* Floating point data element count */
        unsigned long long ac_ve;        /* Vector elements count */
        unsigned long long ac_l1lmc;     /* L1 instruction cache miss count */
        unsigned long long ac_vecc;      /* Vector execution in microseconds */
        unsigned long long ac_l1mcc;     /* L1 cache miss in microseconds */
        unsigned long long ac_l2mcc;     /* L2 cache miss in microseconds */
        unsigned long long ac_ve2;       /* Vector elements count 2 */
        unsigned long long ac_varec;     /* Vector arithmetic execution in microseconds */
        unsigned long long ac_l1lmcc;    /* L1 instruction cache miss in microseconds */
        unsigned long long ac_vldec;     /* Vector load execution in microseconds */
        unsigned long long ac_l1omcc;    /* L1 operand cache miss in microseconds */
        unsigned long long ac_pccc;      /* Port conflict in microseconds */
        unsigned long long ac_ltrc;      /* Load instruction traffic count */
        unsigned long long ac_vlpc;      /* Vector load packet count */
        unsigned long long ac_strc;      /* Store instruction traffic count */
        unsigned long long ac_vlec;      /* Vector load element count */
        unsigned long long ac_vlcme;     /* Vector load cache miss element count */
        unsigned long long ac_vlcme2;    /* Vector load cache miss element count 2 */
        unsigned long long ac_fmaec;     /* Fused multiply add element count */
        unsigned long long ac_ptcc;      /* Power throttling in microseconds */
        unsigned long long ac_ttcc;      /* Thermal throttling in microseconds */
};
```

ac_etime フィールドと ac_utime フィールドの単位は ticks です。Ticks は linux において 10ms を意味しており、ac_etime フィールドの値もしくは ac_utime フィールドの値に 100 を掛けることで秒単位の時間を取得できます。

ac_version フィールドの値は 14 です。構造体が変更になった場合この値は変更されます。

3. 部分的にサポートされているシステムコール

以下は VEOS で部分的にサポートされているシステムコールの一覧です。

SL No	システムコール	相違点 (あり / なし)
1	Clone	あり
2	futex	あり
3	prlimit	あり
4	getrlimit	あり
5	setrlimit	あり
6	wait4	あり
7	clock_nanosleep	あり
8	timer_create	あり
9	getitimer	あり
10	madvise	あり
11	mlock	あり
12	munlock	あり
13	mlockall	あり
14	munlockall	あり
15	setitimer	あり
16	Ioctl	あり
17	exit_group	あり
18	getcpu	あり
19	quotactl	あり
20	set_tid_address	あり
21	Ustat	あり

注意

- 部分的にサポートされている syscalls exit_group()、futex()、getcpu()、quotactl()、set_tid_address()および ustat()は、VEOS の一部として提供される glibc ライブラリによる呼び出しのみサポートされます。 ユーザプログラムによる直接的な呼び出しはサポートされていません。

1. clone()

1. clone()は VEOS で部分的にサポートされています

2. clone()では以下のフラグの組み合わせのみサポートされています。

SL No	フラグ
1.	SIGCHLD
2.	CLONE_PARENT_SETTID | SIGCHLD
3.	CLONE_CHILD_SETTID | CLONE_CHILD_CLEARTID | SIGCHLD
4.	CLONE_VM | CLONE_VFORK | SIGCHLD
5.	CLONE_VM | CLONE_FS | CLONE_FILES | CLONE_SYSVSEM | CLONE_SIGHAND | CLONE_THREAD | CLONE_SETTLS | CLONE_PARENT_SETTID | CLONE_CHILD_CLEARTID | 0

3. 上記に記載されているフラグ以外に、clone()の man ページに記載されているその他のフラグでサポートされているものはありません。clone()はサポートされていないフラグに対して<EINVAL>をリターンします。

4. フラグでは SIGCHILD 以外で指定されているシグナルはありません。その他のシグナルが指定される場合 Point-2、Point-3 で記載されているように clone は<EINVAL>をリターンします。

5. プロセス(メインプロセスを含む)に対して作成されるスレッドの最大値は 64 です。64 以上のスレッドが作成されようとした場合 clone()は<EAGAIN>をリターンします。

6. VEOS でサポートされているスレッドの最大値は 1024 です。1024 以上のスレッドが作成されようとする場合 clone()は<EAGAIN>をリターンします。

7. VEOS での clone()に対する raw システムコールは下記のとおりです。:

```
int clone(int flags, void *stack, pid_t *ptid, pid_t *ctid, void *tls,
void *guard_ptr)
```

このように clone()ラッパ関数の fn と arg 引数は省略されます。

8. clone()は VEOS 特有であり、ポータブルを想定したプログラムで使用されるべきものではありません。

注意

- ❖ Clone() libc ラッパと raw システムコールはライブラリにより使用されるよう想定されています。
- ❖ Clone() libc ラッパと raw システムコールのエンドユーザによる直接使用はサポートされていません。エンドユーザは高水準のシステムコールや fork()、vfork()、pthread_create()など利用可能な API を使用する必要があります。

9. Clone()システムコールは以下のエラーをリターンします。
    a. EAGAIN: プロセスに対し 64 以上のスレッドが作成された場合

b. EAGAIN: (VEOS システム全体で)1024 以上のスレッドが作成された場合
c. EAGAIN: (VEOS システム全体で)256 以上のプロセスが作成された場合

注意
❖ VEOS の RLIMIT_NPROC の取り扱いに対しては getrlimit() / setrlimit()の制限をご参照ください

10. プロセスがオープンファイルディスクリプタの最大限度を使い切った後、clone()システムコールのその後の呼び出しは、errno を EAGAIN に設定し失敗します。
11. chroot()システムコールを使用して呼び出したプロセスのデフォルトのルートディレクトリを path に指定したものに変更した後、clone()の呼び出しは失敗し、errno は EAGAIN に設定されます。

2. futex()

1. futex()は VEOS で部分的にサポートされています。

2. futex()システムコールは以下の futex の機能のみサポートしています。

SL No	フラグ
1.	FUTEX_WAIT
2.	FUTEX_WAKE
3.	FUTEX_REQUEUE
4.	FUTEX_CMP_REQUEUE
5.	FUTEX_WAIT_BITSET
6.	FUTEX_WAKE_BITSET
7.	FUTEX_PRIVATE_FLAG
8.	FUTEX_CLOCK_REALTIME

3. futex マニュアルページで記載されている残りの futex の機能は futex システムコールによりサポートされていません。futex()はサポートされていない futex の全機能に対して<EINVAL>をリターンします。

4. Robust Futex 機能はサポートされていません。通常の futex call のみ上記一覧の機能でサポートされています。

5. 優先度継承 futex (PI-futex)は VEOS ではサポートされていません。

6. futex()は VEOS 固有のものであり、ポータブルを想定したプログラムで使用されるべきではありません。

注意
❖ Libc はこのシステムコールではラッパを提供していません。 ❖ Bare futexes はエンドユーザにとって簡易な概念として想定されていません。

- ❖ Futex のシステムコールのユーザはアセンブリ言語に慣れており、futex のユーザスペースライブラリやカーネルスペースの実装のソースに精通していることが想定されています。
- ❖ プロセスとスレッドの同期やロッキングのために、POSIX セマフォや各種 POSIX スレッド同期のメカニズム(mutexes, condition variables, read-write locks, barriers)を含む futex を通して実行された高水準のプログラミング概念を使用してください。

3. prlimit() / getrlimit() / setrlimit()

1. prlimit()は VEOS で部分的にサポートされています。

2. 以下のフラグは VEOS とは異なった動作となります:

SL No	フラグ	動作
1.	RLIMIT_CPU	マンページごとに、プロセスが CPU 時間を消費し続ける場合、ハードリミット値に達するまで 1 秒に 1 回 SIGXCPU が送信されます。VEOS では、SIGXCPU は 1 回だけ送信されます。
2.	RLIMIT_NPROC	VEOS ではハードリミットとソフトリミットは RLIMIT_NPROC に対し維持されていません。prlimit()は VH ホストカーネルの値を表示し VH ホストカーネルへ値を設定します。 ただし VEOS ではプロセスとスレッドに対して以下の制限があります。 - プロセスの最大値 = 256 -プロセスごとのスレッド = 64 -スレッドの最大値 = 1024 VEOS ではタスク(プロセス/スレッド)の作成中に RLIMIT_NPROC の値は考慮されません。すべての特権プロセス(CAP_SYS_RESOURCE 機能)又は非特権プロセスに対し、制限は上記で定義された値の通りとなります(それぞれ 256,64,1024)。
3.	RLIMIT_NICE	サポートされておらず EINVAL がリターンされます。
4.	RLIMIT_RTPRIO	サポートされておらず EINVAL がリターンされます。

3. pid 1 が引数として与えられる場合、-1 が返され errno に ESRCH が設定されます。

4. wait4()

1. wait4()は VEOS で部分的にサポートされています。

2. 以下のフラグは wait4 ではサポートされていません。

SL No	フラグ
1.	__WCLONE
2.	__WALL

3. wait4 は上記のフラグでサポートされていません。これは”clone”の子が作成できないことによる、clone()システムコールの制限に基づきます。[“clone” child は、シグナルを発信しないもの、または、終了時に親へ SIGCHLD 以外のシグナルを送るものです。]

4. もし子に終了のシグナル(SIGFPE, SIGTERM など)が送られる場合、SIGKILL が実際のシグナルの代わりに送られます。子プロセスが親プロセスで待機する場合、WTERMSIG(status) value = SIGKILL (9)となります。

5. clock_nanosleep ()

1. clock_nanosleep()は VEOS で部分的にサポートされています。

2. clock_nanosleep()では以下のフラグのみサポートされています:

SL No	フラグ
1.	CLOCK_REALTIME
2.	CLOCK_MONOTONIC

3. CLOCK_PROCESS_CPUTIME_ID flag. はサポートされていません。 Clock_nanosleep () は CLOCK_PROCESS_CPUTIME_ID flag に対し EINVAL をリターンします。

6. timer_create()

1. timer_create()は VEOS で部分的にサポートされています。
2. システムコールは以下のフラグのみサポートしています。

SL No	フラグ
1.	CLOCK_REALTIME
2.	CLOCK_MONOTONIC

3. timer_create()マニュアルページで記載されている残りのフラグは timer_create()システムコールでサポートされていません。Timer_create()はすべてのサポートされていないフラグでは EINVAL がリターンされます。

7. getitimer()

1. getitimer()は VEOS で部分的にサポートされています。

2. getitimer()システムコールは以下のフラグのみサポートしています。

SL No	フラグ
1.	ITIMER_REAL

3. getitimer()マニュアルページで記載されている残りのフラグは getitimer ()システムコールでサポートされていません。Getitimer()はサポートされていないフラグに対して EINVAL をリターンします。

8. madvise()

1. VEOS ではページングがサポートされていないため、madvise()システムコールは常に VEOS で成功をリターンします。しかし移植されたアプリケーションは madvise()システムコールを呼び出す可能性があります。

9. mlock()

1. mlock()システムコールは、VEOS でページングがサポートされていないため、常に成功をリターンします。しかし移植されたアプリケーションは mlock()システムコールを呼び出す可能性があります。

10.munlock()

1. VEOS ではページングがサポートされていないため、munlock()システムコールは常に VEOS で成功をリターンします。しかし移植されたアプリケーションは munlock()システムコールを呼び出す可能性があります。

11.mlockall()

1. mlockall()システムコールは、VEOS でページングがサポートされていないため、常に成功をリターンします。しかし移植されたアプリケーションは mlockall()システムコールを呼び出す可能性があります。

12.munlockall()

1. VEOS ではページングがサポートされていないため、munlockall()システムコールは常に VEOS で成功をリターンします。しかし移植されたアプリケーションは munlockall()システムコールを呼び出す可能性があります。

13.setitimer()

1. setitimer()は、VEOS で部分的にサポートされています。

2. setitimer()システムコールは以下のフラグのみサポートしております。

SL No	フラグ
1.	ITIMER_REAL

3. setitimer()マニュアルページにおいて記載されている残りのフラグは setitimer() システムコールによりサポートされていません。Setitimer()はすべてのサポートされていないフラグに対して EINVAL をリターンします。

14.ioctl()

1. ioctl()は VEOS では部分的にサポートされています。
2. VEOS では、ioctl()システムコールを使用した non-tty リクエストは処理されません。 この場合、ioctl()は失敗し、errno は EINVAL に設定されます。

4. サポートされていないシステムコール

以下は VEOS でサポートされていないシステムコールの一覧です。

SL No	システムコール	システムコール呼び出しに伴いリターンされるエラー
1	get_robust_list	ENOTSUP
2	set_robust_list	ENOTSUP
3	unshare	ENOTSUP
4	set_thread_area	ENOTSUP
5	get_thread_area	ENOTSUP
6	prctl	ENOTSUP
7	setpriority	ENOTSUP
8	getpriority	ENOTSUP
9	sched_get_priority_max	ENOTSUP
10	sched_get_priority_min	ENOTSUP
11	sched_setparam	ENOTSUP
12	sched_getparam	ENOTSUP
13	sched_setscheduler	ENOTSUP
14	sched_getscheduler	ENOTSUP
15	clock_settime	EPERM
16	settimeofday	EPERM
17	add_key	ENOTSUP
18	request_key	ENOTSUP
19	keyctl	ENOTSUP
20	reboot	ENOTSUP
21	personality	ENOTSUP
22	sysfs	ENOTSUP
23	setns	ENOTSUP
24	io_setup	ENOTSUP
25	io_destroy	ENOTSUP
26	io_getevents	ENOTSUP
27	io_submit	ENOTSUP
28	io_cancel	ENOTSUP
29	perf_event_open	ENOTSUP
30	ptrace 注釈: ve_ptrace()とは異なります。5 章をご覧ください。	ENOTSUP
31	remap_file_pages	ENOTSUP

32	set_mempolicy	ENOTSUP
33	get_mempolicy	ENOTSUP
34	migrate_pages	ENOTSUP
35	kcmp	ENOTSUP
36	fexit_module	ENOTSUP
37	mremap	ENOTSUP
38	adjtimex	ENOTSUP
39	clock_adjtime	EPERM
40	mbind	ENOTSUP
41	move_pages	ENOTSUP
42	uselib	ENOTSUP
43	_sysctl	ENOTSUP
44	create_module	ENOTSUP
45	get_kernel_syms	ENOTSUP
46	query_module	ENOTSUP
47	nfsservctl	ENOTSUP
48	getpmsg	ENOTSUP
49	putpmsg	ENOTSUP
50	afs_syscall	ENOTSUP
51	tuxcall	ENOTSUP
52	security	ENOTSUP
53	epoll_ctl_old	ENOTSUP
54	epoll_wait_old	ENOTSUP
55	vserver	ENOTSUP
56	swapon	ENOTSUP
57	swapoff	ENOTSUP
58	capget	ヘッダファイル "capability.h"を使用した場合のコンパイルエラー
59	capset	ヘッダファイル "capability.h"を使用した場合のコンパイルエラー
60	vmsplice	ENOTSUP

5. VE Ptrace システムコール

1. VE プログラムでは ptrace()システムコールはサポートされておらず ENOTSUP をリターンします。
2. ptrace()の代わりに ve_ptrace()を呼び出すことにより、VE デバッガのみ ptrace()システムコールを使用できます。
3. トレースされた VE プロセスは、単一のノード上に存在する必要があります。
4. 単一の VE デバッガは、異なる VE ノードの複数の VE プロセスをトレースすることはできません。
5. VEOS デバッガが wait()システムコールファミリから出てくる際に、VE プロセス/スレッドを停止するため、新しい ptrace リクエスト "PTRACE_STOP_VE"は VE デバッガによって呼び出されなければなりません。

```
ve_ptrace(PTRACE_STOP_VE, pid, 0, 0);
```

6. 以下はサポートされていない要求の一覧です。:

SL No	Ptrace リクエスト
1.	PTRACE_SYSEMU
2.	PTRACE_SYSEMU_SINGLESTEP
3.	PTRACE_O_TRACEEXEC
4.	PTRACE_O_TRACEVFORKDONE

7. Ptrace リクエスト PTRACE_GETFPREGS/ PTRACE_SETFPREGS は、ベクトルレジスタを取得/設定します。
8. VE デバッガは、VEOS で特別な処理を行うため、PTRACE_TRACEME を呼び出すことはできません。

6. 一般的な VEOS の相違点/制限

1. FE_DIVBYZERO に対応したトラップが無効である場合、以下の相違点を検討してください。
   子プロセスが整数の 0 による除算(例:5÷0)を実行した場合浮動小数点例外は発生せず子は強制終了されません。これにより wait4(もしくはその他 wait family システムコール)の実行中に子の正確な終了ステータスは親側で受け取られません。

   しかし Linux では、FE_DIVBYZERO に対するトラップが無効であるにもかかわらず、整数の 0 による除算により浮動小数点例外が生じます。そのため子プロセスが上記のような例外により終了した場合、常に期待された終了ステータスがリターンされます。

2. ４つ以上のシグナルを立て続けに受信した時、VE プロセスが異常終了する場合があります。

3. VEOS において、コアパターン(/proc/sys/kernel/core_pattern)が pipe (|)を最初の文字として含有する場合、コアファイルが VE プロセスの現在の作業ディレクトリで作成されます。Pid が xxxx の場合、コアファイルのファイル名は「core.xxxx.ve」となります。

4. VEOS ではコアダンプの作成中は、コアパターンファイルで記載されている場合、“%”, “p”, ”h”のパターンのみサポートされています。これら以外の文字は無視されます。

5. VE プロセスがトレースされていて、ブレークポイントなどを設定した後に si_code を読み込もうとすると、si_code は常に TRAP_BRKPT に設定されます（TRAP_TRACE、TRAP_BRANCH、TRAP_HWBKPT などの SIGTRAP シグナルには si_code が設定されません）。

6. シグナルハンドラが SIGFPE シグナル用に登録されている場合と、ユーザがゼロまたは浮動小数点の除算を 0 で除算した場合の両方の場合に si_code が FPE_FLTDIV に設定されます。Linux の場合、si_code FPE_INTDIV は整数が 0 により除算される場合に設定され、si_code FPE_FLTDIV は浮動小数点が 0 により除算されるときに設定されます。

7. VE プロセスが任意の終了シグナルを受信すると、エンドユーザには、SIGKILL を使用してプロセスが終了したように見えます。これは、すべての終了シグナルに対して、SIGKILL を使用して代理プロセスを終了させるからです。

8. 親プロセスが終了シグナルを子プロセスに送信し、WIFSIGNALED()を使って終了ステータスを取得するのを待つようなマルチプロセス環境では、子プロセスの代理プロセスが SIGKILL によって終了させられ(上記)、WIFSIGNALED() (wait() システムコール)は Linux カーネルにオフロードされるため、親プロセスは常に終了ステータスとして SIGKILL を受け取ります。

9. スタック領域の空きがないためにシグナルハンドラ用のスタックフレームの構築に失敗した場合、VEOS は VE プロセスに対して SIGSEGV を生成し、対応する代理プロセスを SIGKILL で終了させます。この場合、VE プロセスを実行したプログラム(例:シェル)は代理プロセスの終了ステータスを VE プロセスのものとして取得するため、エンドユーザには SIGKILL によって VE プロセスが終了したように見えます。

10. VEOS でシグナル情報がシグナルハンドラ（いくつかの例外のために呼び出される）によって受信されたときに、siginfo 構造体が常に関連する命令アドレス（ICE レジスタ値）を格納する場合、

"si_addr"がファイルされます。しかし Linux では、いくつかの例外のため si_addr は例外が発生した命令のアドレスを格納し、また障害が発生したアドレスを格納します。

VE HW 仕様によると、例外要因によっては、例外を発生させた命令のアドレス、または例外が報告される前に最後に実行された分岐命令のアドレスを保持することがあります。

以下のマスクされていない演算例外の場合、ICE は命令のアドレスを保存して例外を発生させます

- ゼロ除算
- 浮動小数点オーバーフロー例外
- 浮動小数点アンダーフロー例外
- 固定小数点オーバーフロー例外
- 無効な操作例外
- 不正確な例外

例外が発生した場合、ICE は例外が報告される前に最後に実行された分岐命令のアドレスを保存します。

- メモリ保護例外
- 欠落しているページ例外
- スペース不足例外
- メモリアクセス例外
- ホストメモリ保護例外
- ホスト欠落ページ例外
- ホストスペース不足例外
- ホストメモリアクセス例外
- I / O アクセス例外
- 不正なデータフォーマット例外
- 不正な命令フォーマット例外

11. VEOS では、syscalls read() と read64(), futex(),recvfrom(),recvmsg(), recvmmsg(), sendmsg(), sendmmsg(), sendto(), accept(), accept4(), connect()はシグナルの中断後に、自動的に再起動することはありません。

12. VEOS では、VE プロセスが回復不能なハードウェア例外を受信し、VE プロセスがハードウェア例外にマップされたシグナル用のハンドラをインストールしている場合、シグナルハンドラは一度呼び出され、その後 VE プロセスは例外がマップされたシグナルで終了します。
Linux の場合ではプロセスが例外を生成し、ユーザが同じハンドラをインストールした場合、シグナルハンドラが実行された後で原因が発生したのと同じ命令が実行され、シグナルハンドラが無期限に呼び出されます。

13. VEOS ではスレッドが sleep(2) のようなブロッキングシステムコールを実行している場合、pthread_cancel()API を介してスレッド用に生成された SIGCANCEL シグナルは即座には送信されない可能性（スレッドが即座にキャンセル/終了できない）があります。SIGCANCEL の配信は、システムコールのブロッキングなどが実行されるまで延期されます。

14. 同時に VEOS で処理できる要求の最大数は 1056 です。要求は以下により構成されています:
    - VE タスクからの要求(プロセス / スレッド)
    - 移植された RPM コマンドからの要求

- GDB からの要求

注意
❖ VE タスクの最大数= 1024 ❖ VEOS ワーカースレッドの最大数= 1056

15. logging（log4c）が有効になっている場合で、システムコールのように open()/ socket()を使用して割り当てられたとき、VE プロセスは最初のファイル記述子番号として 6 を取得します。3～5 の FD は VEOS 用に予約されています。logging（log4c）が無効である場合で、システムコールのように open()/ socket()を使って割り当てられたとき、VE プロセスは最初のファイル記述子番号として 5 を取得します。3～4 の FD は VEOS 用に予約されています

16. VE アーキテクチャでは、ユーザが任意のタスクの現在の状態を要求または取得しようとする場合、Linux 環境で proc fs インタフェースを使用するのではなく、rpm 固有のコマンドを使用する必要があります。同様に、VE プロセスの実行情報を取得するには、VH rpm コマンドではなく VE 固有 rpm コマンド（ps など）を使用する必要があります。

17. VE アーキテクチャでは、/proc/self ディレクトリを読むことで現在実行中のタスク(self) の情報を得る事はサポートしていません。例えば VE アーキテクチャで、 シンボリックリンク /proc/self/exec にアクセスしても、現在実行中の VE タスクへのパスは返りません。

18. nanosleep / pselect veos が-1 をリターンするように、シグナルハンドラによりシステムコールが中断した場合は、"rem"が NULL でない限り、errno に EINTR を設定し、残りの時間を "rem"が指す構造体に書き込みます。しかし、veos デザインのオフロードとコンテキスト切り替えのオーバーヘッドのために、マイクロ秒単位の "rem"精度は VEOS で異なります。

19. VH が 4KB のページサイズであるのに対して、VE アーキテクチャは、ラージページ（2MB）およびヒュージページ（64MB）をサポートしています。

20. VEOS において、SIGCONT シグナルは SIGKILL や SIGSTOP 同様、マスクできません。sigprocmask(2), sigaction(2), pselect(2)/pselect6(2), ppoll(2), epoll_pwait(2)等のシグナルマスクの更新に関するシステムコールを用いた SIGCONT シグナルへのマスク要求は全て無視され、常に成功します。

21. VE プログラムの set-user-id ビットと set-group-id ビットは無視されます。例えば、たとえ set-user-id ビットがセットされていても、実効ユーザ ID が VE プログラムの所有者に変更されることはありません。同様に、set-group-id ビットがセットされていても、実効グループ ID が VE プログラムのグループに変更されることはありません。

22. VE アーキテクチャにおいて、プログラムは最大で 2GB – 4KB のバッファデータを write(2) や writev(2) のような write 系システムコールによって書くことが出来ます。したがって、VE アーキテクチャの write 系システムコールの返り値の最大は 2GB – 4KB になります。返り値の最大値は VE アーキテクチャのページサイズに依存しません。

23. POSIX 標準によると通常ファイル、パイプ、FIFO への I/O はアトミックである事を意図しています。アトミックであるとは、その他の I/O 命令に割り込まれることなく、 1 つの命令の全てのバイトがまとまって転送を開始し、終了するということです。しかし、VE アーキテクチャは アトミック、非アトミック両方の I/O モードをサポートします。デフォルトでは、VH 側のメモリ消費を抑えるた

めに非アトミックモードが有効になっています。
- アトミック I/O は VE_ATOMIC_IO=1 を設定すると有効になります。

24. VE における VE プロセスと対応する代理プロセスのリソース制限について、ve_exec コマンド、VH の prlimit コマンド、VE の prlimit システムコール、VE に移植された prlimit コマンドの仕様を以下の表に示します。

カテゴリ	**'ve_exec' コマンド**	**VH prlimit コマンド**	**VE prlimit コマンド**	**VE prlimit システムコール**
カテゴリ-1: **FSIZE, LOCKS, MSGQUEUE, NPROC, MEMLOCK, RTTIME, NOFILE**	新しい VE プログラムを実行した時、そのプログラムは対応する代理プロセス (ve_exec) のリソース制限を引き継ぎます。	VE プロセスが実行中に VH の prlimit コマンドで代理プロセスのリソース制限を変更した場合、その値は VE プロセスに反映されます。	VE プロセスが実行中に VE の prlimit コマンドで VE プロセスのリソース制限を変更した場合、その値は 対応する代理プロセスに反映されます。	VE プロセスが実行中に VE の prlimit システムコールで VE プロセスのリソース制限を変更した場合、その値は 対応する代理プロセスに反映されます。
カテゴリ-2: **AS, CPU, CORE, DATA, RSS, SIGPENDING**	新しい VE プログラムを実行した時、そのプログラムは対応する代理プロセスのリソース制限を引き継ぎます。	VE プロセスが実行中に VH の prlimit コマンドで代理プロセスのリソース制限を変更した場合、その値は VE プロセスに反映**されません**。	VE プロセスが実行中に VE の prlimit コマンドで VE プロセスのリソース制限を変更した場合、その値は 対応する代理プロセスに反映**されません**。	VE プロセスが実行中に VE の prlimit システムコールで VE プロセスのリソース制限を変更した場合、その値は 対応する代理プロセスに反映**されません**。
カテゴリ-3: **STACK**	新しい VE プログラムの STACK 制限は "unlimited" か環境変数 "VE_STACK_LIMIT" で指定された値がセットされます。	VE プロセスが実行中に VH の prlimit コマンドで代理プロセスのリソース制限を変更した場合、その値は VE プロセスに反映**されません**。	VE プロセスが実行中に VE の prlimit コマンドで VE プロセスのリソース制限を変更した場合、その値は 対応する代理プロセスに反映**されません**。	VE プロセスが実行中に VE の prlimit システムコールで VE プロセスのリソース制限を変更した場合、その値は 対応する代理プロセスに反映**されません**。

- NICE と PTPRIO のリソース制限は VE ではサポートしていません。
- ulimit、 prlimit コマンド、prlimit システムコールによる RSS の制限は VH において、効力を持ちません。

25. VE のアーキテクチャにおいては、シグナルは VE タスクの状態が RUNNING になり、VE コア上で実行されたときに配送されます。

この動作のため、VE タスクに対して送られたシグナルが保留中の時に、VE タスクが（ブロッキングシステムコールの呼び出しなどで）WAIT 状態に変更されると、シグナルはペンディングキューに残ったままになります。この場合、ブロッキングシステムコールの処理または割り込みの後、タスクの状態が RUNNING に変わると、生成されたシグナルは配送されます。

26. VE プログラムに渡されるコマンド ラインの引数の最大数は 512 です。512 より大きい場合、VE プログラムの実行開始に失敗します。

    VE プログラムに渡される環境変数の最大数は 512 です。512 より大きい場合、VE プログラムの実行開始に失敗します。この環境変数の最大数には、プログラムに常に渡される VE_EXEC_PATH、LOG4C_RCPATH、HOME、PWD、VE_LD_ORIGIN_PATH および VE_NODE_NUMBER の 6 つの環境変数が含まれます。