# HPC System Call Usage Trends

Terry Jones[†], Andrew Tauferner[‡], Todd Inglett[‡]

[†] Lawrence Livermore National Laboratory
P.O. Box 808, L-561
Livermore, CA 94551
trj@llnl.gov

[‡] International Business Machines
IBM Systems & Technology Group
Rochester, MN 55901
{ataufer , tinglett} @us.ibm.com

**Abstract.**

The changing landscape of high performance computing (HPC) hardware and applications poses a constant challenge to operating system architects. In this paper, we assess operating system evolution on the basis of several key factors related to system call functionality. Results are presented for Linux and Linux-like lightweight kernels. Comparisons are made with several other operating systems employed in high performance computing environments including AIX, HP-UX, OpenSolaris, and FreeBSD. Results are provided in tabular form. To our knowledge, this is the first such compilation of system call trends.

## 1. Introduction

A system call is simply the request of an application program for service from the operating system. It is the mechanism through which applications gain access to the operating system kernel – that portion of system software that is permanently memory-resident and responsible for managing system resources such as process scheduling and I/O control. When an application makes a system call, the calling program is making direct use of the facilities provided by the kernel and the major portion of system call code is actually executed as part of the kernel itself and not the calling program. [1] We distinguish system calls from library calls in that library calls may be performed without access to the operating system kernel or its data structures; when a program calls a library call, the code executed is part of the final object program (e.g. common functions such as duplicating a string).

The rapid architectural progress made possible by Moore's law has resulted in exciting platform possibilities with a variety of interesting resources and functional capabilities. For example, the Department of Energy's (DOE's) Advanced Simulation and Computing (ASC) Red Storm machine at Sandia National Laboratory utilizes over 10,000 Opteron-based nodes and a custom ultra high performance 3-D mesh interconnect that services compute nodes as well as service and I/O nodes. [2] The ASC BlueGene/L machine at Lawrence Livermore National Laboratory utilizes system-on-a-chip technology developed for the embedded microprocessor marketplace to reach new levels of performance per rack as well as overall

performance. BlueGene/L utilizes a custom torus and a custom tree interconnect for over 64,000 compute nodes as well as service and I/O nodes. [3] The recently announced Los Alamos National Laboratory RoadRunner system utilizes both Opteron processors as well as Cell chips that are able to handle some processing (the Cell chip was originally designed for the Playstation 3 video game console). [4] Not only does Moore's law result in new and exciting architectures – it is also radically changing our notion of node counts for High Performance Computing (HPC) systems: the size of the most capable machines may range from tens of thousands of nodes to hundreds of thousands of nodes. Such node counts bring new requirements for system wide resource utilization, fault tolerance, and ease of use. These design points and others make clear that system software designers can not assume a homogeneous cluster with a traditional compute node.

Fortunately, there are a number of new efforts underway to assess current and future stress points in the system software stack and to innovate new technologies for HPC platforms. One such effort is the U.S. Department of Energy initiated Fast-OS program to study "fundamental questions in operating system and runtime research that must be explored in order to enable scientific application developers and users to achieve maximum effectiveness and efficiency on this new generation of systems." [5] This paper is the result of one Fast-OS effort, the Colony project, and its attempt to evaluate the current and recent past landscape of HPC system software.

What follows is a record of the progression of kernels in terms of currently provided services, the expansion of kernel services, and sample HPC application usage. The rest of this paper is organized as follows. In Section 2, we distinguish lightweight kernels and full-featured kernels. In Section 3, we provide tabular results for system call trends. Section 4 describes related work. In Section 5, we present our conclusions and our plans for future work. Finally, we provide acknowledgments and references in Section 6.

## 2. Lightweight Kernels and Full-Featured Kernels

Lightweight kernel architectures have a long history of success for highly scalable systems. Examples include the Puma operating system on ASC Red [6], the Catamount operating system on ASC Red Storm, and the CNK operating system on ASC BlueGene/L [7]. Lightweight kernels trade-off the generality derived from a comprehensive set of operating system features for efficiencies in resource utilization, improved scalability for large-scale bulk-synchronous applications, and simplicity of design.

Resource utilization differences between lightweight kernels and Linux-like full-featured operating systems include memory footprint and overhead CPU cycles. Entire lightweight kernels may be 3 MB or less. Overhead CPU cycles are reduced through the removal of various services like those provided by most daemons and those associated with socket communication or asynchronous file I/O.

Studies have shown the services present in full-featured operating systems may introduce interference or jitter. [8] [9] For example, the indirect overhead of periodic operating system (OS) clock interrupts ("ticks") that are used by all general-purpose OSs as a means of maintaining control has been shown to be a second major cause of noise. [10] Furthermore, this interference has been shown to have an impact on common parallel operations such as synchronizing collectives. Synchronizing collective operations are operations in which a set of processes (frequently every process) participates and no single process can continue until every process has participated. Examples of synchronizing collective operations from the MPI interface are MPI_Barrier, MPI_Allreduce, and MPI_Allgather. These operations pose serious challenges to scalability since a single instance of a laggard process will block progress for every other process. Unfortunately, synchronizing collective operations are required for a large class of parallel algorithms and are quite common [11]. A *cascading effect* results when one laggard process impedes the progress of every other process. The cascading effect has significant operating system implications and proves especially detrimental in an HPC context where high processor counts are common. Uncoordinated scheduling of background tasks (such as system daemons) hurts the performance of processes in a job. Today's operating systems lack any awareness of large parallel applications, instead viewing them as thousands of independent processes. Agarwal et al. have presented a theoretical treatise on the effects of interfering jitter on parallel collective operations; they found that certain noise distributions (notably the Pareto distribution and Bernoulli distribution) can dramatically impact performance. [12] Though the impact on a serial job may be small, synchronization in parallel applications amplifies the effect of a local delay and slows down the entire parallel job. These delays can have cascading effects that become the primary scaling bottleneck for the parallel job. [9]

Given the importance of interference and/or jitter to scalability, the fact that lightweight kernel architectures have demonstrated less interference than unmodified full-featured operating systems has resulted in a great deal of interest and research in that architecture for HPC applications. BlueGene/L uses a lightweight kernel called CNK that is optimized for computational efficiency on its compute nodes. To achieve optimal performance this kernel was kept very simple but the designers wanted to maintain a level of familiarity for programmers. To that end a POSIX compliant interface is presented and the glibc runtime library was ported to provide I/O support. In order to maintain CNK's simplicity while providing I/O support, the processing nodes of BlueGene/L are divided into groups of relatively small numbers of compute nodes with an associated I/O node. The compute nodes defer to the I/O nodes for the execution of I/O and other complex system calls. This removes the complexity of a full function I/O software stack from the compute node kernel. A more full function operating system such as Linux running on the I/O nodes can provide complex

functionality without impacting the computational tasks on the compute nodes.

The BlueGene/L compute node kernel provides many system call interfaces. Some of the simpler calls can be handled locally in the compute node kernel while more complex calls must be handled by the I/O node. These complex system calls are shipped to the I/O node for execution. The I/O request and data are shipped over a collective network that joins compute nodes and I/O nodes together into processor sets. A daemon called CIOD runs on the I/O node to receive system call requests from the compute nodes, executing those requests and returning the results. The compute node blocks until the system call result is returned. This design eliminates the need for the various daemons that assist with I/O in many traditional kernels. [13] CNK's lack of daemons, paging activity, and TLB misses due to static TLB mapping ensures that operating system (OS) noise is kept to an absolute minimum.

While the I/O node runs a full Linux kernel, it is also modified for scalability. There is no swap space, a small number of daemons are running, and the root filesystem is completely in memory. Connectivity to file servers and other resources is achieved via a gigabit Ethernet network. The application process does not directly run on the I/O node so security issues inherent in network connectivity are avoided. BlueGene/L's I/O architecture is capable of supporting various high performance parallel filesystems such as GPFS, Lustre, and PVFS2.

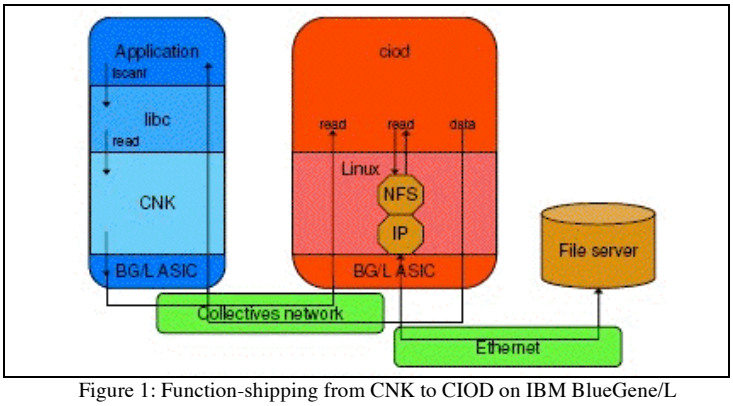


Figure 1: Function-shipping from CNK to CIOD on IBM BlueGene/L

BlueGene/L's separation of responsibilities between compute nodes and I/O nodes leads to a scalable and robust operating system for the compute nodes. The robustness is achieved because the kernel is simple to design, implement, and test. The kernel's scalability is the result of little or no interference with the compute process. The lack of interference has been directly and indirectly measured through operating system noise measurements and other techniques.

The Catamount lightweight kernel also function-ships complex operations to I/O nodes for completion. Catamount, which is utilized on the compute nodes of Sandia's Red Storm system, draws its heritage from earlier work with the Puma lightweight kernel and Cougar lightweight kernel developed by Sandia National Laboratory and the University of New Mexico.

While lightweight kernels offer a jitter-free environment with extremely small levels of operating system interference, they do so at a price. Some system calls are not supported, and there is usually little (if any) support for threaded parallelism. Section 3 includes data on which calls are supported for both Red Storm's Catamount and BlueGene/L's CNK. A full-featured operating system such as Linux would provide a full complement of system calls, but as stated earlier variability in context switch times and interference from normal Linux activities such as daemons have been shown to dramatically reduce scalability for very large node counts. [14] Projects like the Colony Project (http://www.hpc-colony.org) seek to provide scalability in a full-featured Linux operating system while retaining all system calls for worry-free application porting. [15]

## 3. Trends

One of the more tangible ways of tracking the evolution of operating systems is to evaluate their functionality based on system call interfaces. Of necessity, system call interfaces are exposed for both proprietary operating systems and open operating systems. Whereas subtle differences in functional behavior are hard to quantify, assessments based on system calls provide for meaningful comparisons. We evaluate system calls for several important classifications including: system calls that are common to current operating systems employed by HPC environments; available system calls in lightweight kernels and how they are implemented (i.e. performed natively or forwarded to other nodes); system calls that are utilized by representative HPC applications; and how services provided by operating systems are changing over time.

### 3.1 A Common Set of System Calls

In Tables 1a and 1b, we present information on those calls which are found in at least two of the following operating systems: AIX [16], FreeBSD [17], Linux [18], OpenSolaris [19], and HP-UX [20]. The table includes 4 columns: Column 1 labeled Call identifies the system call by name; Column 2 labeled Linux? denotes whether the system call is present in Linux 2.6.18 or not; Column 3 labeled BGL denotes how the call is implemented in the BlueGene/L operating system (CNK indicates in the BlueGene Compute Node Kernel, IO indicates the call is functioned shipped to the IO node); Column 4 labeled Red Storm denotes how the call is implemented in the Red Storm Catamount operating system (Cat indicates Catamount compute node kernel, IO indicates the call is functioned shipped to the IO node, *yod* indicates that the call is functioned shipped to a node running the Red Storm yod daemon).

Table 1a: Lightweight Kernel Implementation of ***Common*** System Calls
Cray-Sandia-UnivNewMexico Catamount and IBM CNK

Each of the following are found in at least two of the following operating systems:
AIX, FreeBSD, Linux, OpenSolaris, HP-UX

| CALL | LINUX? | BGL | Red Stor |
|---|---|---|---|
| _exit | Linux | | |
| accept | Linux | | |
| access | Linux | IO | IO |
| acct | Linux | | |
| adjtime | | | |
| afs_syscall | Linux | | |
| aio_cancel | | | |
| aio_error | | | |
| aio_read | | | |
| aio_return | | | |
| aio_suspend | | | |
| aio_waitcomplete | | | |
| aio_write | | | |
| alarm | Linux | | Cat |
| alloc_hugepages | Linux | | |
| atexit | | | |
| bind | Linux | | |
| brk | Linux | CNK | |
| chdir | Linux | IO | Cat |
| chmod | Linux | IO | IO |
| chown | Linux | IO | IO |
| chroot | Linux | | |
| clock_getres | Linux | | |
| clock_gettime | Linux | | |
| clock_nanosleep | Linux | | |
| clock_settime | Linux | | |
| close | Linux | IO | IO |
| connect | Linux | IO | |
| creat | Linux | IO | IO |
| creat64 | Linux | | |
| dup | Linux | IO | Cat |
| dup2 | Linux | IO | Cat |
| exec | | | |
| execv | | | |
| execve | Linux | CNK | |
| exit | Linux | CNK | yod |
| fchdir | Linux | IO | |
| fchmod | Linux | IO | IO |
| fchown | Linux | IO | IO |
| fcntl | Linux | IO | IO |
| flock | Linux | | |
| fork | Linux | | |
| fpathconf | | | |
| fstat | Linux | IO | IO |
| fstatfs | Linux | IO | IO |
| fsync | Linux | IO | IO |
| ftime | Linux | | Cat |
| ftruncate | Linux | IO | IO |
| getcontext | | | |
| getcwd | Linux | IO | Cat |
| getdents | Linux | IO | IO |
| getdirentries | | | |
| getdomainname | | | |
| getdtablesize | | | |
| getegid | Linux | | Cat |
| geteuid | Linux | | Cat |

| CALL | LINUX | BGL | Red Sto |
|---|---|---|---|
| getgid | Linux | CNK | Cat |
| getgroups | Linux | | |
| gethostid | | | |
| gethostname | Linux | | yod |
| getitimer | Linux | CNK | |
| getlogin | | | |
| getmsg | | | |
| getpagesize | | | |
| getpeername | Linux | IO | |
| getpgid | Linux | | |
| getpgrp | Linux | | |
| getpid | Linux | | Cat |
| getppid | Linux | | |
| getpriority | | | |
| getresgid | Linux | | |
| getresuid | Linux | | |
| getrlimit | Linux | CNK | Cat |
| getrusage | Linux | CNK | Cat |
| getsid | Linux | | |
| getsockname | Linux | IO | |
| getsockopt | Linux | | |
| gettimeofday | Linux | CNK | Cat |
| getuid | Linux | CNK | Cat |
| ioctl | Linux | | IO |
| kill | Linux | CNK | Cat |
| killpg | | | |
| lchown | Linux | IO | |
| link | Linux | IO | IO |
| listen | Linux | | |
| llseek | Linux | IO | IO |
| lockf | | | |
| lseek | Linux | IO | IO |
| lstat | Linux | IO | IO |
| madvise | Linux | | |
| makecontext | | | |
| mincore | Linux | | |
| mkdir | Linux | IO | IO |
| mknod | Linux | | IO |
| mlock | Linux | | |
| mmap | Linux | | |
| mount | Linux | | Cat |
| mprotect | Linux | | |
| msem_init | | | |
| msem_lock | | | |
| msem_remove | | | |
| msem_unlock | | | |
| msgctl | | | |
| msgget | | | |
| msync | Linux | | |
| munlock | Linux | | |
| munmap | Linux | | |
| nanosleep | Linux | | Cat |
| nice | Linux | | |
| open | Linux | IO | IO |
| pathconf | | | |
| pause | Linux | | |

Table 1b: Lightweight Kernel Implementation of ***Common*** System Calls
Cray-Sandia-UnivNewMexico Catamount and IBM CNK

Each of the following are found in at least two of the following operating systems:
AIX, FreeBSD, Linux, OpenSolaris, HP-UX

| CALL | LINUX | BGL | Red Stor |
|---|---|---|---|
| getfh | | | |
| plock | | | |
| poll | Linux | | |
| pread | Linux | | IO |
| profil | Linux | | |
| ptrace | Linux | | |
| putmsg | | | |
| pwrite | Linux | | IO |
| quotactl | Linux | | |
| read | Linux | IO | IO |
| readdir | Linux | | IO |
| readlink | Linux | IO | IO |
| readv | Linux | IO | IO |
| reboot | Linux | | |
| recv | Linux | IO | |
| recvfrom | Linux | IO | |
| recvmsg | Linux | | |
| rename | Linux | IO | IO |
| rmdir | Linux | IO | IO |
| rtprio | Linux | | |
| sbrk | Linux | | |
| sched_get_priority_max | Linux | | |
| sched_get_priority_min | Linux | | |
| sched_getparam | Linux | | |
| sched_getscheduler | Linux | | |
| sched_rr_get_interval | Linux | | |
| sched_setparam | Linux | | |
| sched_setscheduler | Linux | | |
| sched_yield | Linux | | |
| select | Linux | | |
| semctl | | | |
| semget | | | |
| semop | | | |
| send | Linux | IO | |
| sendfile | Linux | | |
| sendmsg | | | |
| sendto | Linux | IO | |
| setdomainname | Linux | | |
| setgid | Linux | | yod |
| setgroups | Linux | | |
| sethostname | Linux | | |
| setitimer | Linux | CNK | Cat |
| setpgid | Linux | | |
| setpgrp | Linux | | |
| setpriority | Linux | | |
| setregid | Linux | | |
| setresgid | Linux | | |
| setresuid | Linux | | |
| setreuid | Linux | | |
| setrlimit | Linux | CNK | Cat |
| setsid | Linux | | |
| setsockopt | Linux | | |
| settimeofday | Linux | | |

| CALL | LINUX | BGL | Red Stor |
|---|---|---|---|
| pipe | Linux | | |
| setuid | Linux | | yod |
| shmctl | | | |
| shmdt | | | |
| shmget | | | |
| shutdown | | | |
| sigaction | Linux | CNK | Cat |
| sigaltstack | Linux | | |
| sigblock | | | |
| signal | Linux | CNK | Cat |
| sigpause | | | |
| sigpending | Linux | | Cat |
| sigprocmask | Linux | | Cat |
| sigreturn | Linux | CNK | |
| sigsetmask | Linux | | |
| sigstack | | | |
| sigsuspend | Linux | | |
| sigvec | | | |
| socket | Linux | IO | |
| socketpair | Linux | | |
| stat | Linux | IO | IO |
| statfs | Linux | IO | IO |
| stime | Linux | | |
| stty | Linux | | |
| swapon | Linux | | |
| swapoff | Linux | | |
| symlink | Linux | IO | IO |
| sync | Linux | | |
| sysconf | | | |
| sysfs | Linux | | |
| syslog | Linux | | |
| time | Linux | CNK | Cat |
| times | Linux | CNK | |
| truncate | Linux | IO | IO |
| ualarm | | | |
| ulimit | Linux | | |
| umask | Linux | IO | Cat |
| umount | Linux | | Cat |
| uname | Linux | CNK | Cat |
| unlink | Linux | IO | IO |
| unmount | Linux | | |
| ustat | Linux | | |
| utime | Linux | IO | IO |
| utimes | Linux | | |
| vfork | Linux | | |
| vhangup | Linux | | |
| wait | Linux | | |
| wait3 | | | |
| wait4 | Linux | | |
| waitid | Linux | | |
| waitpid | Linux | | |
| write | Linux | IO | IO |
| writev | Linux | IO | IO |

### 3.2 System Calls That are Unique To Linux

In Tables 2a and 2b, those calls that are unique to Linux are presented. That is, each call is found in Linux but not in AIX, FreeBSD, OpenSolaris, or HP-UX. As in the previous section this table includes 4 columns: Column 1 labeled Call identifies the system call; Column 2 labeled BGL denotes how the call is implemented in the BlueGene/L OS; and Column 3 labeled Red Storm denotes how the call is implemented in the Red Storm Catamount OS.

Table 2a: Lightweight Kernel Implementation of ***Linux-Only*** System Calls
Cray-Sandia-UnivNewMexico Catamount and IBM CNK

Each of the following are found in Linux but not the following operating systems: AIX, FreeBSD, OpenSolaris, HP-UX

| CALL | BGL | Red Storm | CALL | BGL | Red Storm |
|---|---|---|---|---|---|
| add_key | | | get_thread_area | | |
| adjtimex | | | getdents64 | IO | |
| bdflush | | | getegid32 | | |
| break | | | geteuid32 | | |
| capget | | | getgid32 | | |
| capset | | | getgroups32 | | |
| chown32 | | | getpmsg | | |
| clone | | | getresgid32 | | |
| creat6 | | | getresuid32 | | |
| create_module | | | gettid | | |
| delete_module | | | getuid32 | | |
| epoll_create | | | getxattr | | |
| epoll_ctl | | | gtty | | |
| epoll_wait | | | init_module | | |
| exit_group | | | inotify_add_watch | | |
| faccessat | | | inotify_init | | |
| fadvise64_64 | | | inotify_rm_watch | | |
| fchmodat | | | io_cancel | | |
| fchown32 | | | io_destroy | | |
| fchownat | | | io_getevents | | |
| fcntl64 | | | io_setup | | |
| fdatasync | IO | IO | io_submit | | |
| fgetxattr | | | ioperm | | |
| flistxattr | | | iopl | | |
| free_hugepages | | | ioprio_get | | |
| fremovexattr | | | ioprio_set | | |
| fsetxattr | | | ipc | | |
| fstat64 | IO | | kexec_load | | |
| fstatat64 | | | keyctl | | |
| fstatfs64 | IO | | lchown32 | | |
| ftruncate64 | | | lgetxattr | | |
| futex | | | linkat | | |
| futimesat | | | listxattr | | |
| get_kernel_syms | | | llistxattr | | |
| get_mempolicy | | | lock | | |

Table 2b: Lightweight Kernel Implementation of ***Linux-Only*** System Calls
Cray-Sandia-UnivNewMexico Catamount and IBM CNK

Each of the following are found in Linux but not the following operating systems:
AIX, FreeBSD, OpenSolaris, HP-UX

| CALL | BGL | Red Storm |
|---|---|---|
| get_robust_list | | |
| lremovexattr | | |
| lsetxattr | | |
| lstat64 | IO | |
| mbind | | |
| migrate_pages | | |
| mkdirat | | |
| mknodat | | |
| mlockall | | |
| mmap2 | | |
| modify_ldt | | |
| move_pages | | |
| mprotect | | |
| mpx | | |
| mq_getsetattr | | |
| mq_notify | | |
| mq_open | | |
| mq_timedreceive | | |
| mq_timedsend | | |
| mq_unlink | | |
| mremap | | |
| munlockall | | |
| newselect | | |
| nfsservctl | | |
| oldfstat | | |
| oldstat | | |
| openat | | |
| personality | | |
| pivot_root | | |
| ppoll | | |
| prctl | | |
| prof | | |
| pselect6 | | |
| putpmsg | | |
| query_module | | |
| readahead | | |
| readlinkat | | |
| remap_file_pages | | |
| removexattr | | |
| renameat | | |
| request_key | | |
| rt_sigaction | | |
| rt_sigpending | | |
| rt_sigprocmask | | |
| rt_sigqueueinfo | | |
| rt_sigreturn | | |
| rt_sigsuspend | | |
| rt_sigtimedwait | | |

| CALL | BGL | Red Storm |
|---|---|---|
| lookup_dcookie | | |
| sched_getaffinity | | |
| sched_setaffinity | | |
| security | | |
| sendfile64 | | |
| set_mempolicy | | |
| set_robust_list | | |
| set_thread_area | | |
| set_tid_address | | |
| setfsgid | | |
| setfsgid32 | | |
| setfsuid | | |
| setfsuid32 | | |
| setgid32 | | |
| setgroups32 | | |
| setregid32 | | |
| setresgid32 | | |
| setresuid32 | | |
| setreuid32 | | |
| setuid32 | | |
| setxattr | | |
| sgetmask | IO | |
| socketcall | | |
| ssetmask | | |
| stat64 | IO | |
| statfs64 | IO | |
| symlinkat | | |
| sync_file_range | | |
| sysctl | | |
| sysinfo | | |
| tee | | |
| tgkill | | |
| timer_create | | |
| timer_delete | | |
| timer_getoverrun | | |
| timer_gettime | | |
| timer_settime | | |
| tkill | | |
| truncate64 | IO | |
| umount2 | | |
| unlinkat | | |
| unshare | | |
| uselib | | |
| vm86 | | |
| vm86old | | |
| vmsplice | | |
| vserver | | |
| prctl | | |

### 3.3 System Call Growth

As of version 2.6.18, Linux has 313 system calls. This represents a growth of about 60% since Linux version 2.4.2 (see Figure 2). Of the 313 system calls found in Linux, 46 are uniform in name, parameters, and functionality to FreeBSD, NETBSD, OPENBSD, BEOS, and ATHEOS operating systems. [21]

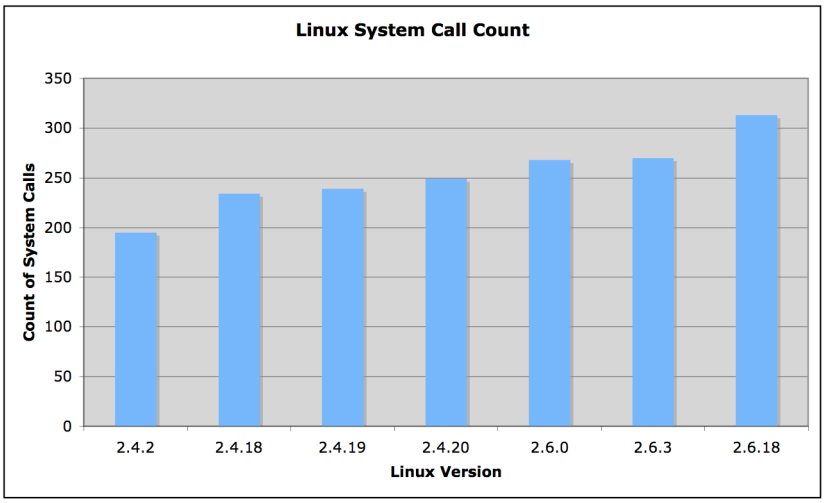


Figure 2: Growth of Linux System Call Count

The reason for this growth is varied. System calls have been added for clarifying 32-bit or 64 bit data structures, improving performance of context switches [22], expanding parallel library support to include Java and OpenMP [23], adding support for emerging standards such as POSIX real-time system calls, and addressing new application functionality demands.

It is interesting to compare today's Linux to the pre-TCP Version 6 Unix operating system of 1977. John Lion's teaching materials on V6 Unix listed 215 "operating system procedures" (which includes calls like sched and panic), 43 of which were listed as "system call entries". [24] [25]

### 3.4 System Call Usage Among HPC Applications

In Tables 3a and 3b, we identify 7 HPC applications and/or commonly used libraries and note which system calls each application uses. The applications were selected based on their availability and popularity. Popularity was determined by ad hoc usage at representative supercomputer centers (especially the supercomputer center at Lawrence Livermore National Laboratory which is familiar to the authors).

A script was developed to parse C source files, header files, and FORTRAN source files for application usage of system calls. The script produces a log that must be interpreted by a programmer to ensure that the call usage should be included.

The columns to Tables 3a and 3b have the following meanings: column 1 lists the name of a system call; column 2 (labeled "BGL") indicates calls that are supported on BlueGene/L with a blue background; column 3 (labeled) "Red") indicates calls that are supported on Red Storm; and columns 4 through 9 list the seven HPC applications (Co-op, NAMD, ARES, ALE3D, BLAST, POP, X11).

Table 3a: HPC Application Usage of System Calls

| | BGL | Red | Co-op | NAMD | ARES | ALE3D | BLAST | POP | libX11 |
|---|---|---|---|---|---|---|---|---|---|
| accept | | | ✓ | ✓ | ✓ | ✓ | ✓ | | |
| bind | | | ✓ | ✓ | ✓ | ✓ | ✓ | | |
| chdir | | | | ✓ | ✓ | | | | |
| close | | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| connect | | | | ✓ | | ✓ | ✓ | | |
| creat | | | | | | | ✓ | | |
| dlcose | | | ✓ | | | | | | |
| dlerror | | | ✓ | | ✓ | | | | |
| dlopen | | | ✓ | | ✓ | ✓ | | | ✓ |
| dlsym | | | ✓ | | ✓ | ✓ | | | |
| dup2 | | | | | | | ✓ | | |
| execv | | | | | | | ✓ | | |
| execve | | | | | ✓ | | | | |
| exit | | | | | ✓ | ✓ | ✓ | | |
| fcntl | | | | | ✓ | ✓ | ✓ | | |
| fork | | | | | ✓ | | ✓ | | |
| fstat | | | | | | | ✓ | | ✓ |
| getcontext | | | | ✓ | ✓ | | | | |
| getcwd | | | | | ✓ | ✓ | | | |
| getgroups | | | | | ✓ | | | | |
| gethostname | | | | ✓ | ✓ | ✓ | ✓ | | |
| getitimer | | | | ✓ | | | | | |
| getpeername | | | ✓ | | | | ✓ | | |
| getpid | | | | | ✓ | ✓ | ✓ | | |
| getpwuid | | | | | ✓ | ✓ | ✓ | | |
| getrlimit | | | | | | | ✓ | | |
| getrusage | | | | ✓ | | ✓ | | | |
| getsockname | | | ✓ | | | | | | |
| getsockopt | | | | | | | ✓ | | |
| gettimeofday | | | ✓ | ✓ | | ✓ | | | |
| getuid | | | | | ✓ | | | | ✓ |
| ioctl | | | | | | ✓ | | | |
| kill | | | | | | | ✓ | | |
| lchown | | | | | | | | | |
| lgetxattr | | | | | | | | | |
| link | | | | | | ✓ | | | |
| linkat | | | | | | | | | |
| listen | | | ✓ | | ✓ | ✓ | ✓ | | |
| lseek | | | | | ✓ | | | | |
| madvise | | | | | | | ✓ | | |
| mkdir | | | | | ✓ | ✓ | | | |
| mmap | | | | ✓ | | | ✓ | | |

Table 3b: HPC Application Usage of System Calls

| | BGL | Red | Co-op | NAMD | ARES | ALE3D | BLAST | POP | libX11 |
|---|---|---|---|---|---|---|---|---|---|
| mprotect | | | | ✓ | | | | | |
| open | | | | ✓ | ✓ | ✓ | | ✓ | |
| pipe | | | | | | | ✓ | | |
| poll | | | | | | | ✓ | | ✓ |
| read | | | ✓ | | ✓ | ✓ | ✓ | ✓ | |
| readdir | | | | | ✓ | ✓ | ✓ | | |
| recv | | | | | ✓ | ✓ | ✓ | | |
| recvfrom | | | | ✓ | | | ✓ | | |
| rename | | | | ✓ | ✓ | ✓ | | | |
| rmdir | | | | | ✓ | | | | |
| sbrk | | | | ✓ | | | | | |
| select | | | | ✓ | ✓ | ✓ | ✓ | | |
| send | | | | ✓ | ✓ | ✓ | | | |
| sendto | | | | | | | ✓ | | |
| setcontext | | | | ✓ | ✓ | | | | |
| setrlimit | | | | | | | ✓ | | |
| setsockopt | | | | | | ✓ | ✓ | | |
| shmat | | | | | | | | | ✓ |
| shmctl | | | | | | | | | ✓ |
| shmdt | | | | | | | | | ✓ |
| sigaction | | | | | ✓ | ✓ | ✓ | | |
| signal | | | | | ✓ | | ✓ | | |
| sleep | | | | | ✓ | ✓ | | | ✓ |
| socket | | | ✓ | ✓ | ✓ | ✓ | ✓ | | |
| stat | | | | | ✓ | | ✓ | | |
| symlink | | | | | ✓ | ✓ | | | |
| system | | | | | ✓ | ✓ | ✓ | | |
| time | | | ✓ | | ✓ | | ✓ | | |
| times | | | | | ✓ | ✓ | | | |
| uname | | | | | | | ✓ | | ✓ |
| unlink | | | | | ✓ | ✓ | | | ✓ |
| usleep | | | ✓ | | | | ✓ | | |
| vfork | | | | | ✓ | | | | |
| waitpid | | | | | ✓ | | | | |
| write | | | ✓ | | ✓ | ✓ | ✓ | ✓ | ✓ |
| writev | | | | | | | ✓ | | |

Cooperative parallelism (Co-op) is a new programming model being developed at Lawrence Livermore National Laboratory. It supports task parallelism by allowing applications to spawn and remotely invoke "symponents," which encapsulate executing sequential or data parallel programs. A runtime system known as Coop supports the creation and management of these symponents, and it appears to the operating system and the resource manager as an MPI parallel application. [26]

NAMD is a parallel molecular dynamics code designed for high-performance simulation of large biomolecular systems. Based on Charm++ parallel objects, NAMD scales to hundreds of processors on high-end parallel platforms and tens of processors on commodity clusters using gigabit ethernet. NAMD won the 2002 Gordon Bell prize. [27]

ARES is a large three-dimensional (3D) physics code developed at LLNL. ARES uses regular grid parallel domain decomposition and explicit MPI. [28]

ALE3D is a large three-dimensional (3D) multi-physics code developed at LLNL. It uses an arbitrarily connected hexahedral element mesh, and each timestep involves a lagrange step with a mesh remap and subsequent advection of materials through the relaxed mesh. This hydrodynamic algorithm is referred to as "Arbitrary Lagrange-Eulerian", or ALE. ALE3D also supports explicit or implicit time integration, interaction between discontinuous portions of the mesh (slide surfaces), advanced material modeling capabilities, chemistry, thermal transport, incompressible flow, and an impressive set of 3rd party solver libraries to support various matrix-based algorithms. [29]

BLAST is a popular genomics application. BLAST (Basic Local Alignment Search Tool) provides a method for rapid searching of nucleotide and protein databases. Since the BLAST algorithm detects local as well as global alignments, regions of similarity embedded in otherwise unrelated proteins can be detected. Both types of similarity may provide important clues to the function of uncharacterized proteins. [30]

POP is an ocean circulation model derived from earlier models of Bryan, Cox, Semtner and Chervin in which depth is used as the vertical coordinate. The model solves the three-dimensional primitive equations for fluid motions on the sphere under hydrostatic and Boussinesq approximations. Spatial derivatives are computed using finite-difference discretizations that are formulated to handle any generalized orthogonal grid on a sphere, including dipole and tripole grids that shift the North Pole singularity into land masses to avoid time step constraints due to grid convergence. The code is written in Fortran90. [31]

The X Window System is a graphics system primarily used on Unix systems (and, less commonly, on VMS, MVS, and MS-Windows systems) that provides an inherently client/server oriented base for displaying windowed graphics. The X Window System provides a public protocol by which client programs can query and update information on X servers. LibX11 is the library used by clients and servers to build graphical user interfaces with X capabilities. [32]

## 4. Related Work

While there is a wealth of material to support various design decisions for individual operating system designs, to our knowledge this is the first paper that analyzes system call trends with the emphasis on operating design and application usage.

Within the last decade, a significant body of work has been developed on operating system call usage in support of Intrusion Detection Systems (IDs). Intrusion detection systems build models of expected behavior and

identify possible attacks based on deviations from the established model. On host-based systems, system call patterns are frequently incorporated in the model to characterize normal behavior. IDs may be used to detect malicious behavior directed towards batch resources, the network, and other computational resources. [33] [34] [35]

Another topic with some similarities is operating system protection domains. Long time Unix aficionados have no doubt been exposed to viewing the operating system in terms of a series of concentric circles with the hardware being at the innermost core circle and applications being at the outermost circle. Most system calls do make use of special protection which prevents accidental or malicious access misuse. [36] However, protection design may be separated from system call design. In fact, the UNIX operating systems has been implemented in User Space. [37]

## 5. Conclusions and Future Work

New requirements unmet by traditional operating system call sets are surfacing with the new crop of computer architectures and new levels of node counts. In some instances, it is sufficient to add new system calls to meet these new needs and Linux has swelled 60% between 2.4.2 and 2.6.18. Other needs such as scalability to tens of thousands of nodes are not addressed so easily and novel kernels are being developed for these unique needs such as the lightweight kernel.

A catalogue of system calls supported by todays most successful lightweight kernels reveals 83 of the 218 system calls found in common HPC full-featured operating systems are supported (either natively or by function-shipping to a remote node). Finally, an evaluation of 7 applications/libraries reveals references to 78 system calls, 45 of which are satisfied by lightweight kernels. Each of these 7 application/libraries could probably be ported to lightweight kernels with good results, but to do so might require more effort than the typical new platform port.

A desirable operating system would provide all system calls of interest to HPC applications while retaining excellent scalability. The Colony project is researching novel strategies to enable full-featured Linux to scale to hundreds of thousands of nodes with excellent scalability. We are continuing to investigate the needs of HPC applications and to research strategies for improving scalability and reduced runtime variability.

## 6. Acknowledgements and References

This work benefits from the helpful contributions of Suzanne Kelly from Sandia National Laboratory, and Gary Kumfert, Mike Collette and Rob Neely from Lawrence Livermore National Laboratory, and Gengbin Zheng and Celso Mendes from the University of Illinois at Urbana-Champaign, and Albert Sidelnik of IBM.

Funding for this work was provided by the U.S. Department of Energy Office of Science under the Fast-OS program. This work was performed under the auspices of the U.S. Department of Energy by University of California Lawrence Livermore National Laboratory under contract No. W-7405-Eng-48.

### References

1. Keith Haviland, Ben Salama. Unix System Programming. Addison Wesley Publishing Company. 1987.
2. ASC http://www.sandia.gov/ASC/Red Storm.html (ASC Red Storm)
3. ASC http://www.llnl.gov/computing_resources/asc/bluegenel/ (ASC BlueGene/L)
4. ASC http://www-03.ibm.com/press/us/en/pressrelease/20210.wss (ASC Road Runner)
5. Fast-OS http://www.cs.unm.edu/~fastos/
6. S. R. Wheat, A. B. Maccabe, R. E. Riesen, D. W. van Dresser, and T. M. Stallcup, *"PUMA: An Operating System for Massively Parallel Systems,'"* Journal of Scientific Programming, vol. 3, no.4, Winter 1994, (special issue on operating system support for massively parallel systems) pp 275-288.
7. George Almasi, Ralph Bellofatto, Calin Cascaval, Jose G. Castanos, Luis Ceze, Paul Crumley, C. Christopher Erway, Joseph Gagliano, Derek Lieber, Jose E. Moreira, Alda Sanomiya, and Karin Strauss. *"An Overview of the BlueGene/L System Software Organization"* Proceedings of the International Conference on Parallel and Distributed Computing (Euro-Par 2003) Aug. 2003, Klagenfurt, Austria.
8. Shawn Dawson, Mike Collette, "Performance of Ares," LLNL Technical Report UCRL-PRES-152041, January 21, 2003.
9. Terry Jones, Shawn Dawson, Rob Neely, William Tuel, Larry Brenner, Jeff Fier, Robert Blackmore, Pat Caffrey, Brian Maskell, Paul Tomlinson, Mark Roberts, *"Improving the Scalability of Parallel Jobs by adding Parallel Awareness to the Operating System"*. Proceedings of Supercomputing 2003, Phoenix, AZ, November 2003.
10. Tsafrir, D., Etsion, Y., Feitelson, D. G., and Kirkpatrick, S. 2005. *"System noise, OS clock ticks, and fine-grained parallel applications."* In Proceedings of the 19th Annual international Conference on Supercomputing (Cambridge, Massachusetts, June 20 - 22, 2005). ICS '05. ACM

Press, New York, NY, 303-312. DOI= http://doi.acm.org/10.1145/1088149.1088190

11. A. Gupta, A. Tucker, and S. Urushibara, "*The Impact of Operating System Scheduling Policies and Synchronization Methods on the Performance of Parallel Applications*," In Proceedings of the 1991 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems, pages 120-132, May 1991.
12. S. Agarwal, R. Garg, and N. K. Vishnoi. "*The impact of noise on the scaling of collectives: A theoretical approach*," In Proceedings of the 12th International Conference on High Performance Computing, volume 3769 of *Springer Lecture Notes in Computer Science*, pages 280–289, Goa, India, Dec. 2005.
13. IBM Journal of Research and Development, volume 49, Number 2/3, March/May 2005
14. Terry Jones, *"A Scaling Investigation on IBM SPs",* ScicomP 6, Aug. 2002, Berkeley, CA.
15. Chakravorty, S., Mendes, C. L., Kalé, L. V., Jones, T., Tauferner, A., Inglett, T., and Moreira, J. 2006. HPC-Colony: services and interfaces for very large systems. SIGOPS Oper. Syst. Rev. 40, 2 (Apr. 2006), 43-49. DOI= http://doi.acm.org/10.1145/1131322.1131334
16. IBM Technical Reference: Base Operating System and Extensions, Volume 1 and 2. http://inetsd01.boulder.ibm.com/doc_link/en_US/a_doc_lib/libs/basetrf1/basetrf1tfrm.htm
17. FreeBSD http://www.freebsd.org/
18. Linux http://www.linux.org/
19. OpenSolaris http://www.opensolaris.org/os/
20. HP-UX http://docs.hp.com/en/oshpux11.0.html
21. Peter Recktenwald's Linux System Call webpage http://www.lxhp.in-berlin.de/lhpsysc0.html
22. E. Zadok, S. Callanan, A. Rai, G. Sivathanu, and A. Traeger. 2005. Efficient and Safe Execution of User-Level Code in the Kernel. In Proceedings of the 19th IEEE international Parallel and Distributed Processing Symposium (Ipdps'05) - Workshop 10 - Volume 11 (April 04 - 08, 2005). IPDPS. IEEE Computer Society, Washington, DC, 221.1. DOI= http://dx.doi.org/10.1109/IPDPS.2005.189
23. G.D. Benson, M. Butner, S. Padden, and A. Fedosov. The Virtual Processor Interface: Linux Kernel Support for User-level Thread Systems. Proceeding of The IASTED Conference on Parallel and Distributed Computing and Systems Dallas, Texas, USA November 13-15, 2006.
24. J. Lions. A Commentary On The Sixth Edition UNIX Operating System. Department of Computer Science, The University of New South Wales. 1977.
25. J. Lions. UNIX Operating System Source Code Level Six. Department of Computer Science, The University of New South Wales. 1977.

26. Cooperative Parallelism Project (Co-op) http://www.llnl.gov/casc/coopParallelism/
27. NAMD http://www.ks.uiuc.edu/Research/namd/
28. D. Post. Codes Written by the National and International Computational Physics Community. Technical Report LA-UR-02-6284. pp 91-92. www.highproductivity.org/026284coverCEEGcodes.pdf
29. R. Couch, R. Sharp, I. Otero, J. Hsu, R. Neely, S. Futral, E. Dube, T. Pierce, R. McCallen, J. Maltby, and A. Nichols, "ALE Hydrocode Development," Joint DOD/DOE Munitions Technology Development Progress Report, UCRL-ID-103482-97, FY97.
30. BLAST ftp://ftp.ncbi.nlm.nih.gov/blast/temp/ncbi.tar.gz
31. POP http://climate.lanl.gov/Models/POP/POP_2.0.1.tar.Z
32. X Window System http://ftp.x.org/pub/X11R7.1/src/everything/libX11-X11R7.1-1.0.1.tar.gz
33. João B. D. Cabrera, Lundy Lewis, Durham, and Raman K. Mehra. "*Detection and classification of intrusions and faults using sequences of system calls*". ACM SIGMOD Record archive Volume 30 , Issue 4 (December 2001) pp. 25 - 34 ISSN:0163-5808
34. D. Mutz, F. Valeur, G. Vigna, and C. Kruegel. 2006. Anomalous system call detection. ACM Trans. Inf. Syst. Secur. 9, 1 (Feb. 2006), 61-93. DOI= http://doi.acm.org/10.1145/1127345.1127348
35. Gao, D., Reiter, M. K., and Song, D. 2004. Gray-box extraction of execution graphs for anomaly detection. In Proceedings of the 11th ACM Conference on Computer and Communications Security (Washington DC, USA, October 25 - 29, 2004). CCS '04. ACM Press, New York, NY, 318-329. DOI=http://doi.acm.org/10.1145/1030083.1030126
36. Maurice Bach. The Design of the Unix Operating System. Prentice-Hall, Inc., Englewood Cliffs, New Jersey. 1986. Pp. 4-6.
37. David Golub, Randall Dean, Alessandro Forin, and Richard Rashid. Unix as an application program. In Proceedings of the Summer 1990 USENIX Conference, pages 87--95, June 1990.