Adding a System Call to Plan 9

John Floren (john@csplan9.rit.edu)

Sandia National Laboratories
Livermore, CA 94551

DOE/NNSA Funding Statement

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

SAND number 2008-2232W

Adding a System Call

The process for adding a new system call to Plan 9 is rather simple. For this example, a "kernel getpid " function will be added, mirroring the functionality of the getpid() function, but using a system call rather than a libc function.

There are four files that must be modified to add a new system call:

/sys/src/libc/9syscall/sys.h
/sys/include/libc.h
/sys/src/9/port/systab.h
One of /sys/src/9/port/[sysauth, sysfile, sysproc, sysseg].c, depending on the syscall type

/sys/src/libc/9syscall/sys.h (User-mode and Kernel-mode)

The first file that will be modified is /sys/src/libc/9syscall/sys.h. This file numbers the system calls; by default, the last syscall listed is pwrite , index 51. We will add #define KGETPID 52 to the end of the file. When libc is built, it generates a set of small assembly functions that move the system call number to a register and perform interrupt 0x40; now that kgetpid has been added, a function will be generated that moves 52 to the register and does the interrupt.

/sys/include/libc.h (User-mode)

It is necessary to modify /sys/include/libc.h to define kgetpid. The appropriate line in this case is extern int kgetpid(void);. Failing to insert this definition will result in a compiler warning.

/sys/src/9/port/systab.h (Kernel-mode)

Next, the new system call must be registered in /sys/src/9/port/systab.h ; this file contains an array of the system call functions. When a system call interrupt is generated, the correct function is located in the array and called; it then performs the desired operation and returns, allowing the system to go back to user mode. A script, /sys/src/9/port/mksystab, will create a new systab.h file automatically; simply run rc /sys/src/9/port/mksystab > /sys/src/9/port/systab.h

Examining the new file, a new line containing Syscall syskgetpid; is now among the rest of the Syscall definitions, and two new entries in the systab[] and sysctab[] arrays have been added, containing [KGETPID] syskgetpid, in systab[] and [KGETPID] Kgetpid , in sysctab[]. This means that when a system call interrupt is generated with an argument of 52 (the index of kgetpid in the array), the trap handler will access systab[KGETPID] and call the handler function, syskgetpid.

/sys/src/9/port/[sysauth, sysfile, sysproc, sysseg].c (Kernel-mode)

Finally, the handler function, syskgetpid, must be written. Since kgetpid is a process-related function, /sys/src/9/port/sysproc.c is the appropriate file to modify. The syskgetpid function is exceptionally simple:

```
long
syskgetpid(ulong *arg)
{
        return up->pid;
}
```

Examples of more complex functions are in the sysproc.c file.

Compiling and Testing

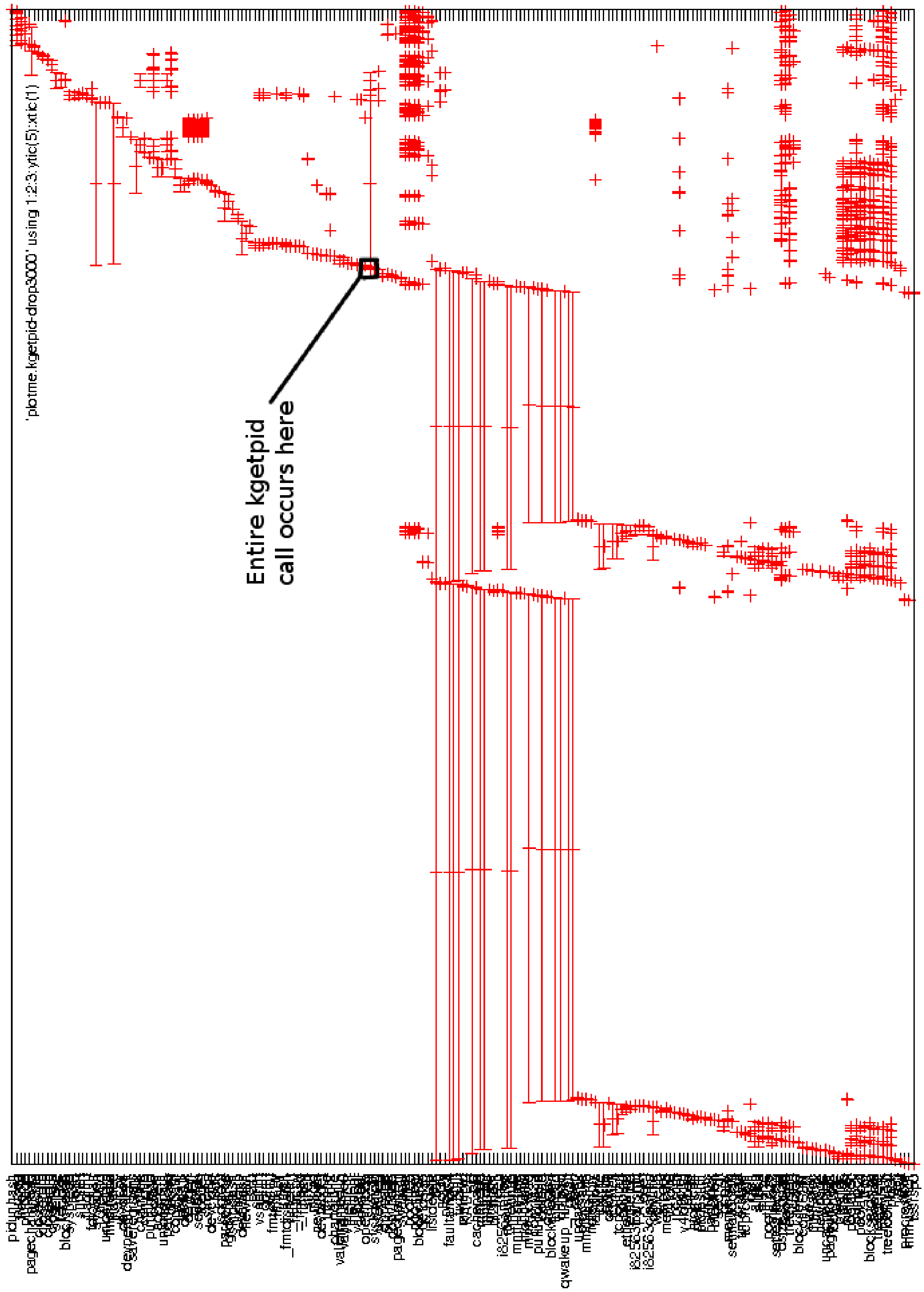
The new kernel is now ready to be built. Since libc was modified, rebuild libc first, then build the kernel as usual. Now, a test program will work as expected:

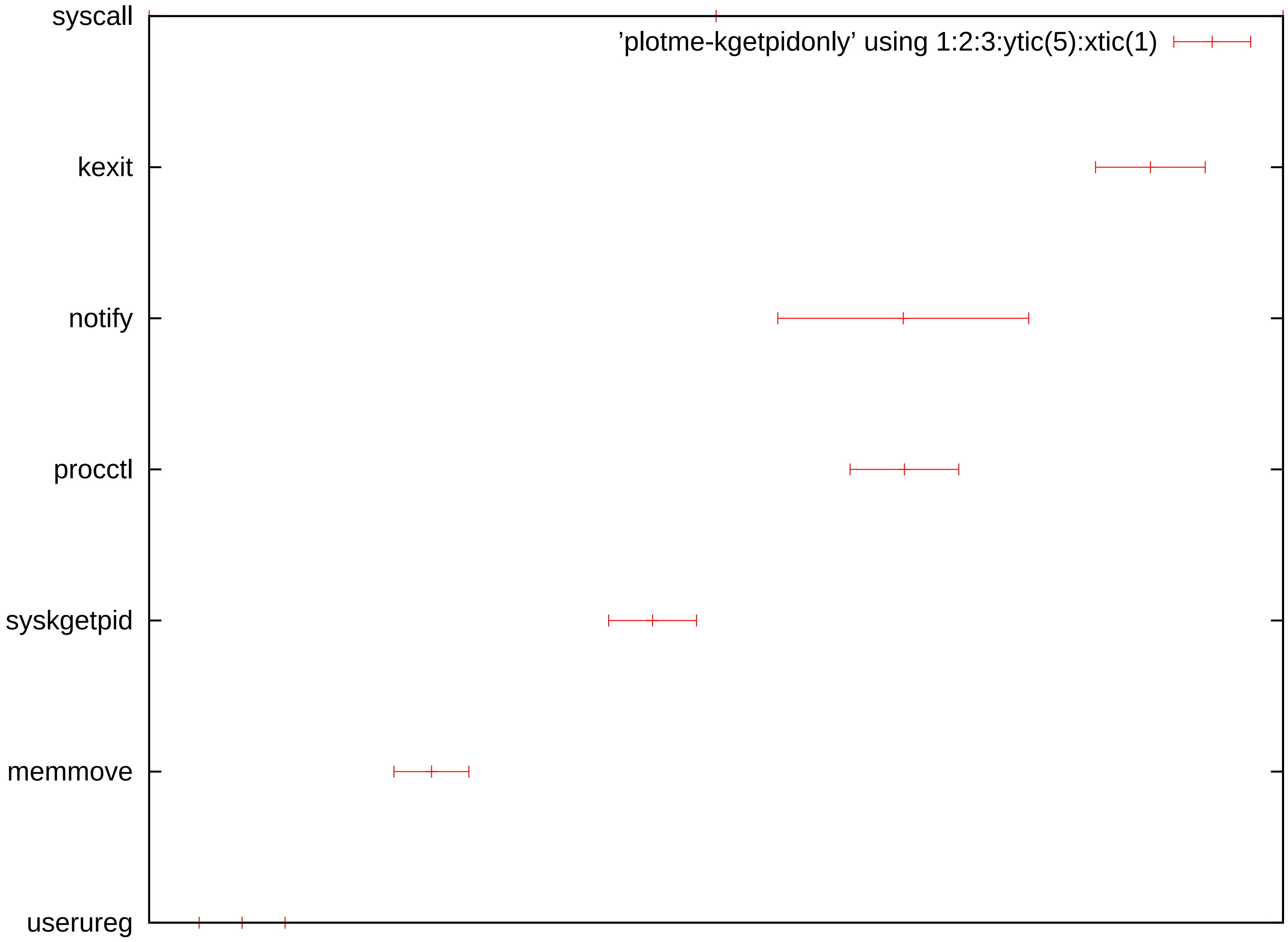
```
% cat > kgetpid.c
#include <u.h>
#include <libc.h>

void main() {
        print("My pid: %d\n", kgetpid());
        exits(0);
}
^D
% 8c kgetpid.c; 8l kgetpid.8; 8.out
My pid: 123
```

Tracing the New System Call

Using a kernel tracing tool still in development, it is possible to examine the relative amount of time spent in executing the system call. The test program is changed as shown below; data is then collected with the tracing tool and plotted. The plot on the next page shows the results gathered; since many functions were called during the execution of the program, it is difficult to read the function labels on the y-axis, but a box has been placed around the area of time where the `kgetpid` system call ran. The execution time for the system call is minimal compared to the process startup and shutdown overhead. The second plot shows the area within the box; as the graph shows, the actual process of executing a syscall is very simple.g4

```
#include <u.h>
#include <libc.h>

void main() {
        kgetpid();
        exits(0);
}
```


'plotme.kgetpid-drop3000' using 1:2:3:ytic(5):xtic(1)
Entire kgetpid
call occurs here


syscall
kexit
notify
procctl
syskgetpid
memmove
userureg
'plotme-kgetpidonly' using 1:2:3:ytic(5):xtic(1)