

Implementing system calls in FreeBSD OS

Shreshth Tuli - Udit Jain

Dept. of Computer Science, IIT Delhi

8 Feb 2019

Outline

- 1 Installing FreeBSD
- 2 HelloWorld! system call
 - Kernel modifications
 - System call implementation
 - Calling system call from C code!
- 3 Debugging
 - Errors generated in buildworld
 - Qemu unparallelisable → Virtual Manager
- 4 Further Comments

Installing FreeBSD

In a FreeBSD VM to install from source,

- `svn update /usr/src`
- `cd /usr/src`

Installing FreeBSD

In a FreeBSD VM to install from source,

- `svn update /usr/src`
 - `cd /usr/src`
 - `make -j6 buildworld`
 - Compiles the tools and other utilities needed in FreeBSD.
- Doesn't have to be re-run subsequently if we're only changing the kernel files.

Installing FreeBSD

In a FreeBSD VM to install from source,

- `svn update /usr/src`
- `cd /usr/src`
- `make -j6 buildworld`
- `make -j6 kernel`
 - Compiles the kernel. And set's it to be loaded as the default kernel.

Installing FreeBSD

In a FreeBSD VM to install from source,

- `svn update /usr/src`
- `cd /usr/src`
- `make -j6 buildworld`
- `make -j6 kernel`
- `shutdown -r now`

Compile time

buildworld took > 24 hours while using single core!

Kernel Modifications

- Added system call ID and function interface to the end
/kern/syscalls.master
- ```
546 AUE_NULL STD {int helloworld();}
```
- Definition for system call, it's prototype, that would be copied to definitions in .h files during make

# Kernel Modifications

- Added system call ID and function interface to the end  
/kern/syscalls.master  
546 AUE\_NULL STD {int helloworld();}
- `sudo make -C sys/kern/ sysent`
  - This changes dir to `sys/kern/` and then makes 'sysent' target there, and calls a script to fetch all sys\_calls from `syscalls.master` to put it into different configuration files.



# Kernel Modifications

- Added system call ID and function interface to the end  
/kern/syscalls.master  
546 AUE\_NULL STD {int helloworld();}
- `sudo make -C sys/kern/ sysent`
- Added to the file /sys/conf/files  
kern/sys\_helloworld.c standard
  - This is to recognize our new file kern/sys\_helloworld.c as a valid file and to compile it in make runs.

# Kernel Modifications

- Added system call ID and function interface to the end  
/kern/syscalls.master  
546 AUE\_NULL STD {int helloworld();}
- sudo make -C sys/kern/ sysent
- Added to the file /sys/conf/files  
kern/sys\_helloworld.c standard
- Added the system call to /kern/capabilities.conf  
##  
## HelloWorld system call by Shreshth and Udit  
##  
helloworld  
-List of system calls enabled in capability mode(Process mode  
in which access to global OS name-spaces is restricted, only  
explicitly declared memory mappings, calls can be used.)

# Kernel Modifications

- Added system call ID and function interface to the end  
/kern/syscalls.master

```
546 AUE_NULL STD {int helloworld();}
```

- `sudo make -C sys/kern/ sysent`

- Added to the file /sys/conf/files

```
kern/sys_helloworld.c standard
```

- Added the system call to /kern/capabilities.conf

```
##
```

```
HelloWorld system call by Shreshth and Udit
```

```
##
```

```
helloworld
```

- `sudo make -j6 kernel`  
`sudo reboot`

# System Call Implementation

In the file kern/sys\_helloworld.c, define the implementation

```
1 ...
2 #ifndef _SYS_SYSPROTO_H_
3 struct helloworld_args {};
4 #endif
5
6 int sys_helloworld(struct thread *td, struct
 helloworld_args *args)
7 {
8 printf("Hello World!\n");
9 return 0;
10 }
11 ...
```

This method of passing arguments is better than in xv6. We don't have to worry about types of variables and extract the appropriate arguments from the stack.

# Calling system call from C code!

```
1 #include <sys/syscall.h>
2 #include <unistd.h>
3
4 int main(){
5 __asm__("mov $564, %rax\n\t"
6 "syscall\n\t");
7
8 }
```

# Calling system call from C code!

- Specify the syscall by writing the number into the eax(32 bit)/rax(64 bit) register.
- Call the interrupt using int \$0x80 or syscall and the kernel performs the task
- The return / error value of a syscall is written to eax/rax
- Parameters are passed by setting the general purpose registers as following:

| Syscall | Param 1 | Param 2 | Param 3 | Param 4 | Param 5 | Param 6 |
|---------|---------|---------|---------|---------|---------|---------|
| rax     | rdi     | rsi     | rdx     | r10     | r8      | r9      |

- Return value in rax

# Undefined type error

```
...
/usr/kh/freebsd/sys/sys/signal.h:297:4: error: unknown
type name 'int32_t'; did you mean '__int32_t'?
int32_t __spare1__;
~~~~~__int32_t  
...  
5 errors generated.
```

We defined the types.h file in the required file(signal.h), it ran. But we're not so sure why was it required.

# Qemu unparallelisable

We first installed FreeBSD on qemu virtualizer, but it couldn't use >1 core even after testing out different CL args. And consequently took >24 hours for buildworld(I stopped it in 24).

So we tried different virtualizers (for both correct number of cores and multiprocessor compatibility) and virt-manager works best with qcow2 images.



## Further Comments

- Implemented reverse copy system call in FreeBSD
- Read different paging algorithms but not sure on how much to change. Changing fundamentally or just optimizing here and there.
- Currently reading the book : The design and implementation of FreeBSD operating system.