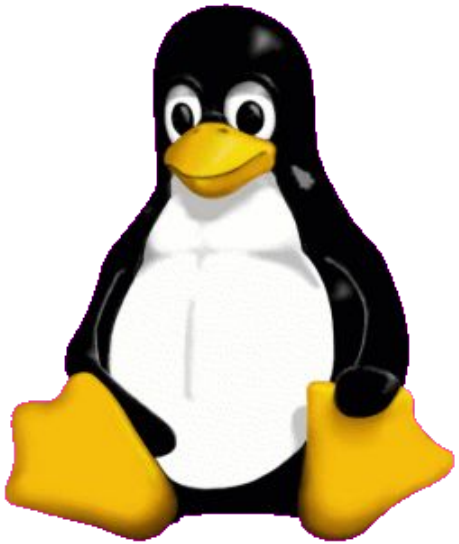
Assignment 2

Creating a Custom System Call



Boot Process of a Linux System

1. **User-Space Request**
   - User-space program calls read() with appropriate arguments.
   - This read function is not an OS feature, but is rather provided by glibc.
2. **Glibc**
   - Sets up the proper arguments according to the architecture.
   - Calls the syscall instruction.
3. **Kernel Invocation**
   - The read() system call is intercepted by the kernel.
   - The kernel processes the call, verifies arguments, and performs the operation.
4. **Kernel Operation**
   - The kernel reads data from the file descriptor and stores it in the provided buffer.
5. **Return to User-Space**
   - The result (number of bytes read) is returned to the user-space program.
   - Errors are communicated via return values and errno.

Syscall Code

```c
#include <unistd.h>

int main() {
    char buf[1024];
    // read from stdin
    read(0, buf, 1024);
    return 0;
}
```

Syscall Code

```c
#include <unistd.h>
#include <sysdep-cancel.h>

/* Read NBYTES into BUF from FD.  Return the number read or -1.  */
ssize_t
__libc_read (int fd, void *buf, size_t nbytes)
{
  return SYSCALL_CANCEL (read, fd, buf, nbytes);
}
```

Syscall Code

```
// Source: https://code.woboq.org/userspace/glibc/sysdeps/unix/sysv/linux/read.c.html
//         (and subsequent includes)

ssize_t __libc_read (int fd, void *buf, size_t nbytes) {
    //Edited for brevity

    unsigned long int resultvar;

    size_t __arg3 = nbytes;
    void* __arg2 = buf;
    int __arg1 = fd;

    register size_t _a3 asm ("rdx") = __arg3;
    register void* _a2 asm ("rsi") = __arg2;
    register int _a1 asm ("rdi") = __arg1;

    asm volatile (
          "syscall\n\t"
        : "=a" (resultvar)
        : "0" (__NR_read), "r" (_a1), "r" (_a2), "r" (_a3)
        : "memory", "cc", "r11", "cx"
    );

    if ((unsigned long int)(resultvar) >= -4095 L) {
      __set_errno(-(resultvar));
      resultvar = (unsigned long int) - 1;
    }

    return resultvar;
}
```

Thank You!