System Call Programming

Week 7

Processor Modes

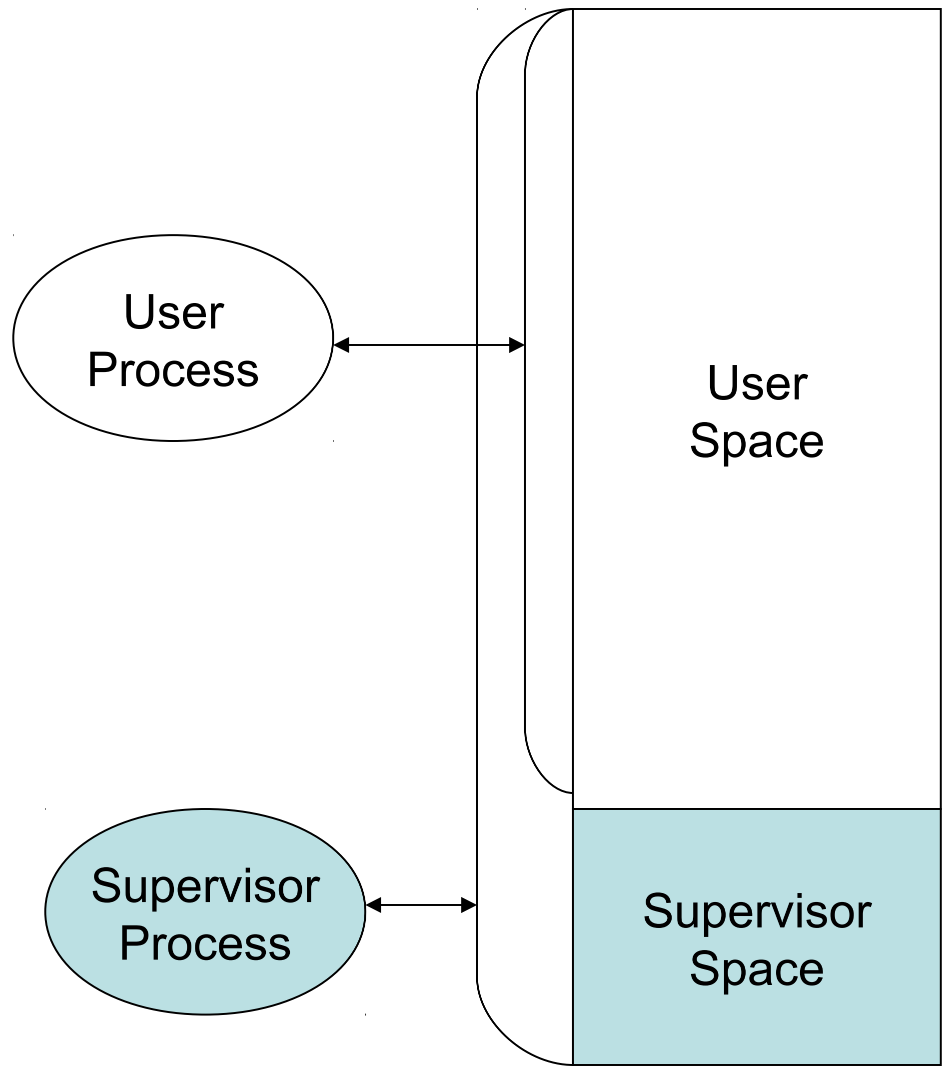
- Mode bit used to distinguish between execution on behalf of OS & execution on behalf of user.
- **Supervisor mode**: processor executes every instruction in its hardware repertoire.
- **User mode**: can only use subset of instructions

Processor Modes

- Instructions that can be executed in supervisor mode are supervisor privileges, or protection instruction
  - **I/O instructions** are protected. If an application needs to do I/O, it needs to get the OS to do it on its behalf.
  - Instructions that can change the **protection state** of the system are privileges (e.g., process' authorization status, pointers to resources, etc)

Processor Modes

- Mode bit may define areas of memory to be used when the processor is in supervisor mode vs user mode



The Kernel

- Core of OS software **executing in supervisor** state
- **Trusted software:**
  - Manages hardware resources (CPU, Memory and I/O)
  - Implements protection mechanisms that could not be changed through actions of untrusted software in user space
- **System call interface** is a **safe way** to expose privileged functionality and services of the processor

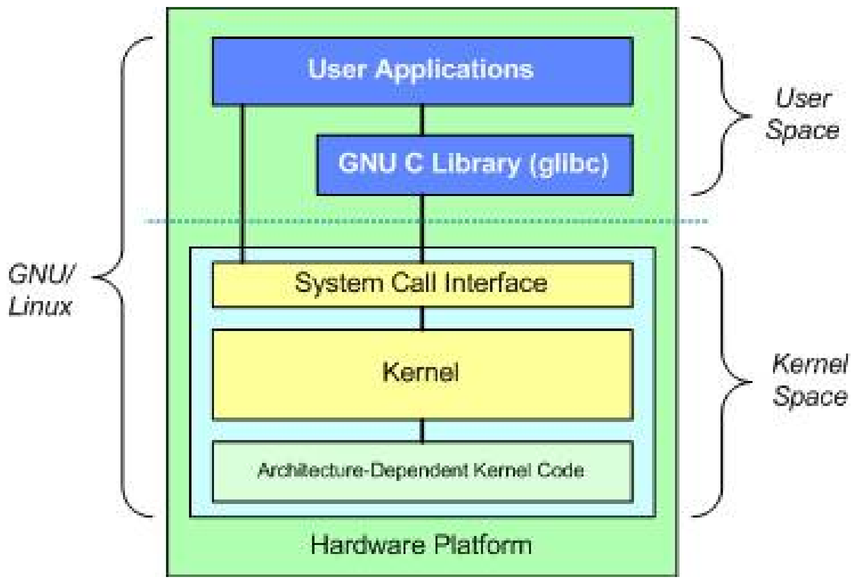
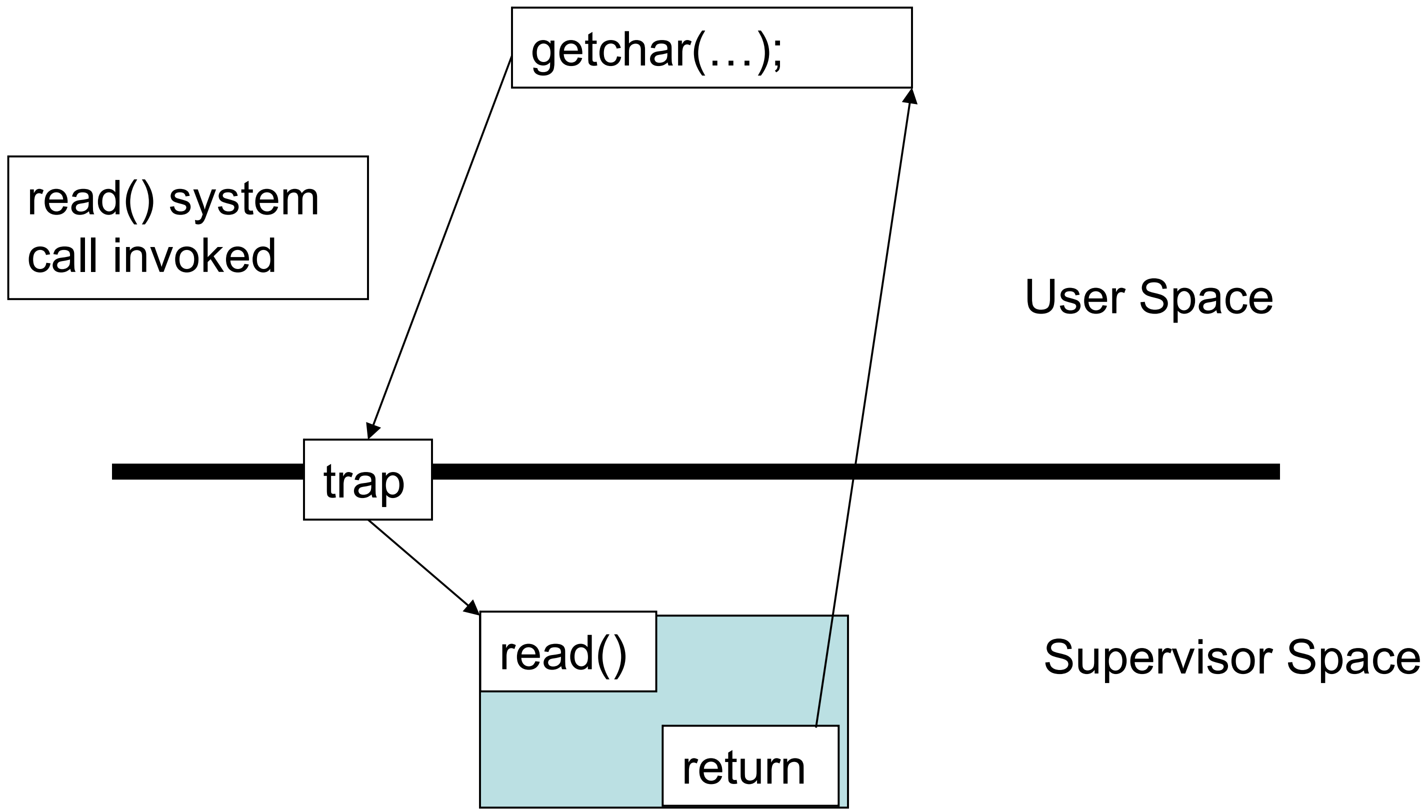


Image by: Tim Jones (IBM)

System Calls



Trap: System call causes a switch from user mode to kernel mode

System calls

- A system call involves the following
  - The system call causes a 'trap' that interrupts the execution of the user process (user mode)
  - The kernel takes control of the processor (kernel mode/privilege switch)
  - The kernel executes the system call on behalf of the user process
  - The user process gets back control of the processor (user mode/privilege switch)
- System calls have to be used **with care**. Expensive due to **privilege switching**.

System Calls

`#include<unistd.h>`

- `ssize_t` **`read`**`(int fildes, void *buf, size_t nbyte)`
  - fildes: file descriptor
  - buf: buffer to write to
  - nbyte: number of bytes to read
- `ssize_t` **`write`**`(int` *`fildes`*`, const void *`*`buf`*`, size_t` *`nbyte`*`);`
  - fildes: file descriptor
  - buf: buffer to write from
  - nbyte: number of bytes to write
- `int` **`open`**`(const char *pathname, int flags, mode_t mode);`
- `int` **`close`**`(int fd);`
- File descriptors
  - 0 stdin
  - 1 stdout
  - 2 stderr

Why are these system calls and not just regular library functions?

Example System Calls

- `pid_t **getpid**(void)`
  - Returns the process ID of the calling process
- `int **dup**(int fd)`
  - Duplicates a file descriptor fd. Returns a second file descriptor that points to the same file table entry as fd does.
- `int **fstat**(int filedes, struct stat *buf)`
  - Returns information about the file with the descriptor filedes into buf

```
struct stat {
    dev_t     st_dev;     /* ID of device containing file */
    ino_t     st_ino;     /* inode number */
    mode_t    st_mode;    /* protection */
    nlink_t   st_nlink;   /* number of hard links */
    uid_t     st_uid;     /* user ID of owner */
    gid_t     st_gid;     /* group ID of owner */
    dev_t     st_rdev;    /* device ID (if special file) */
    off_t     st_size;    /* total size, in bytes */
    blksize_t st_blksize; /* blocksize for file system I/O */
    blkcnt_t  st_blocks;  /* number of 512B blocks allocated */
    time_t    st_atime;   /* time of last access */
    time_t    st_mtime;   /* time of last modification */
    time_t    st_ctime;   /* time of last status change */
};
```

Laboratory

- Write `tr2b` and `tr2u` programs in 'C' that transliterates bytes. They take two arguments 'from' and 'to'. The programs will transliterate every byte in 'from' to corresponding byte in 'to'
  - `./tr2b 'abcd' 'wxyz' < bigfile.txt`
    - Replace 'a' with 'w', 'b' with 'x', etc
  - `./tr2b 'mno' 'pqr' < bigfile.txt`
- `tr2b` uses **getchar** and **putchar** to read from STDIN and write to STDOUT.
- `tr2u` uses **read** and **write** to read and write each byte, instead of using getchar and putchar. The nbyte argument should be 1 so it reads/writes a single byte at a time.
- Test it on a big file with 5000000 bytes

Laboratory

- read and write are system calls.
- getchar and putchar are library functions that live in user space but call read and write. And do buffering!
  - getchar() calls read() only if the local buffer is empty
  - getchar() reads multiple bytes into its buffer in one call to read()

Buffering Issues

- What is buffering?
- Why do we buffer?
- Can we make our buffer really big?

strace and time

- **strace**: Intercepts and prints out system calls to stderr or to an output file.

```
strace -o strace_output ./tr2u 'a' 'b' < bigfile.txt
```

- **time**: Timing

```
time ./tr2b 'x' 'y' < bigfile.txt
```

```
real    0m0.150s
```
(Total time elapsed)

```
user    0m0.132s
```
(time spent by CPU executing in user space)

```
sys     0m0.016s
```
(time spent by CPU executing in kernel space)

Buffered versus Unbuffered

- How many read and write calls are there in each?
- Why does one take longer than the other?

Writing to a File vs. Terminal

- Do you see any differences in the read/writes?
  - `./tr2u 'a' 'b' < bigfile.txt              (TERMINAL)`
  - `./tr2u 'a' 'b' < bigfile.txt > out.txt  (FILE)`
- In the buffered version, how many bytes are being read/written when outputting to a file and when outputting to the terminal?
- Why is there a difference?
- What are the performance considerations?

Homework

- Rewrite sfrob.c using system calls
- If stdin is a regular file, your program initially allocate enough memory to hold all data in the file all at once.
  - Otherwise, it should behave like before
- In addition to the functions you used last time, you'll need **read**, **write**, and **fstat**
  - Read the man pages for these functions
- The program should be named **sfrobu.c**
- Measure differences in performance between **sfrobu** and **sfrob** using the **time** command.
- Write a shell script **sfrobs** that uses standard 'tr' and 'sort' command to sort encrypted files using a pipeline (no temporary files should be created!). Use the 'time' command to compare its performance to **sfrob** and **sfrobu**.