Reminders

- Project 1 due tomorrow at 5:00 pm
- Homework 2 out, due next Wednesday
- Start thinking about project groups (3 people) for the rest of the quarter
- Today: project 1 questions

Project 1 – issues

- C strings
- Copy_to/from_user and counters
- Syscalls: macros ; arguments
- Execvp
- Other things

C strings

- You only need to use:
  - `strncmp(src,dest,256)` – compare strings, 0 if equal, not 0 o.w.
  - `strtok`:
    - 1st use: `tok = strtok(buf, \"delimiters\")`;
    - Subsequent uses: `tok = strtok(NULL, \"delimiters\")`;
  - `fgets`(`buf, 256, stdin`) – read a line (up to 256 chars) from stdin (or getline)
  - `(maybe) strncpy(dest, src, 256)` – copy up to 256 chars from src to dest
  - `(maybe) Allocate memory with malloc, free with free`
- Fine to assume:
  - A maximum length for a shell command (say, 256)
  - Maximum number of arguments (say, 256 again)

Passing counters

- Do not printk the statistics in execcounts!!!
- Execcounts should pass count values to the shell
  - The shell then prints out statistics
- Copying counters to userspace:
  - Shell passes in something to hold data
    - This could be a pointer to a struct, an array, or four pointers to ints.
  - execcounts fills the data in

Copying data to/from kernel

- Unsafe to directly access user pointers!
  
```c
long sys_gettimeofday(struct timeval *tv)
{
    struct timeval ktv;
    do_gettimeofday(&ktv);
    if (copy_to_user(tv, &ktv, sizeof(ktv)))
        return -EFAULT;
    return 0;
}
```

- Copy_to/from_user return amount of uncopied data

Syscalls

- Two ways to use one:
  - Linux style:
    ```c
    #define __NR_foo 292
    static inline __syscall2(int, foo, int, arg1, char *, arg2)
    ```
    - In userspace, just call `foo(4,"test");`
  - BSD style:
    ```c
    #define __NR_foo 292
    ret = syscall(__nr_foo, arg1, arg2);
    ```
### How syscalls work

In entry S:

```
ENTRY(system_call)
  push %eax # save orig_eax
  save_all
  cmp 1 (%nr_syscalls,%eax)
  jae badsys
  call *SYMBOL_NAME(sys_call_table)(%eax,4)
  movl %eax,%esp(4) # save the return value
restore_all:
  RESTORE_ALL
```

### Execvp

- You must build an array of strings to pass to it
- Make sure the last thing in this array is NULL!
- Make sure the array includes the program name!

### Extern

How do we access global variables defined in one file from another file?

```c
char *buffer;
buffer = malloc(100);
strcpy(buffer, param);
```

### Code quality

- What’s wrong with this:
  ```c
  char * buffer;
  buffer = malloc(100);
  strcpy(buffer, param);
  ```
  - How do we fix this?

### Debugging

```c
#define MYDEBUG
#endif

#ifdef MYDEBUG
  #define DEBUG(x) x
#else
  #define DEBUG(x)
#endif

... DEBUG(printf("debug output"));
```

### Debugging 2

- Just for printing:
  ```c
  #ifdef MYDEBUG
  ifdef __KERNEL__
    #ifdef _DEBUG
      /* This one if debugging is on, and kernel space */
      define DEBUGLOG(fmt,...) printk("%mgpr: %fmt", #fmt, #args)
    else
      /* This one for user space */
      define DEBUGLOG(fmt,...) fprintf(stderr, fmt, ...])
    endif
  endif
  #endif
  #endif
  ```
  - works for both for kernel and userspace
  
- To use:
  ```c
  DEBUGLOG("Testing two numbers: %d and %d\n", num, num2);
  ```
Things to check

- Check that every malloc has a matching free
- Check for errors
  - E.g. execvp returns -1, malloc returns NULL
  - Frequently, global constant errno will be set
  - Use perror("error description"); to see what the error was.

Fork

- How does it work?
- Any problems with it?

UNIX startup

- Kernel only creates one process: init (pid 1)
  - Never creates any other processes
- init spawns other processes using fork, exec, etc.
- E.g. init creates a bunch of getty processes
  - One for each terminal
  - Each outputs "login: " prompt, gets input, execs login process
    - Login prompts for a password, execs shell if it's ok
- Off-topic: why is cd a built-in command in your shell?

A UNIX Process

<table>
<thead>
<tr>
<th>Memory</th>
<th>PCB</th>
</tr>
</thead>
<tbody>
<tr>
<td>State Data</td>
<td>Stack Data</td>
</tr>
<tr>
<td>Stack</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Text</td>
<td>Heap</td>
</tr>
</tbody>
</table>

Same Process after fork