CSCI 3753: Operating Systems Fall 2024

Dylan Sain

Department of Computer Science

University of Colorado Boulder

Week 2: Pointers in C

Memory in C

- How does a computer know where to find the value of a variable?

- Memory address
  - `Hexadecimal vs decimal`
  - `0x_____`

C Pointers

- A pointer is a variable whose value is the address of another variable.

- Pointer variable declaration

```
type *var-name;
```

  - For example:
    - `int *ip;` /* pointer to an integer */
    - `double *dp;` /* pointer to a double */
    - `float *fp;` /* pointer to a float */
    - `char *ch;` /* pointer to a character */

C Pointers

- To get the address of a variable: use operator **&**
  - `int var = 20;`
  - `printf("Address of var variable: %x\n", &var);`

- To assign the address of a variable to a pointer
  - `int var;`
  - `int *ip;`
  - **`ip = &var;`**

- To access the value at the address available in the pointer variable: use operator *****
  - `printf("Value of var variable via ip pointer: %d\n", *ip);`

Function Call By Reference

- The call by reference method is to pass the address of variables to arguments of a function.
- To pass a value by reference, argument pointers are passed to the functions just like any other value.

Exercise 1: Swap two numbers using pointers

- `int *ip;` : pointer declaration
- `&ip` : variable address
- `*ip` : pointer value

Function Call By Reference

- For example

```
/* function definition to swap the
values */

void swap(int *x, int *y) {
    int temp;
    temp = *x; /* save the value at address x */
    *x = *y; /* put y into x */
    *y = temp; /* put temp into
    y */
    return;
}
```

Function Call By Reference

```
#include <stdio.h>

/* function declaration */
void swap(int *x, int *y);

int main () {
    /* local variable definition */
    int a = 100;
    int b = 200;

    /* calling a function to swap the values. * &a
    indicates pointer to a ie. address of variable a and *
    &b indicates pointer to b ie. address of variable b. */
    swap(&a, &b);

    printf("After swap, a=%d and b=%d\n", a, b);
    return 0;
}
```

Arrays & Pointers

- An array can be defined with a given fixed size.
  - If the size defined is smaller than is needed, there is not enough memory to hold all of the elements.
  - If the size defined is larger than is needed, memory is wasted.

- Solution?

  Define the array as a dynamic variable using pointer

Fixed-size definition

```
type arrayName[arraySize];
```

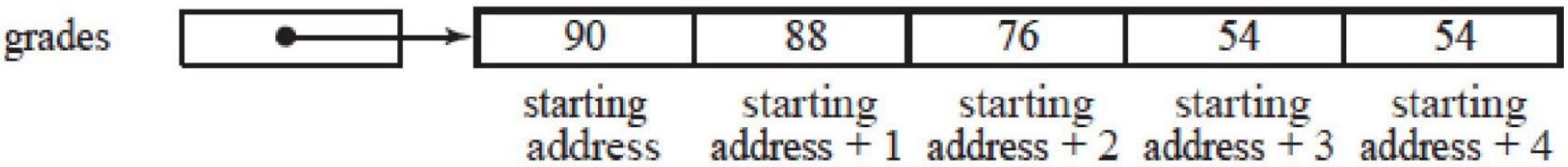
Dynamic-allocation definition

```
type *arrayName;
arrayName = (cast-type*) malloc(byte-size);
...
free(arrayName);
```

Trivia: Why do arrays start at location 0?

Arrays & Pointers

- When passed to functions, arrays are passed by reference.
- An array name is a pointer to the beginning of the array.
- Access of an individual element of an array through an index is done by pointer arithmetic.




```
cout << grades[0] << *grades

cout << grades[2] << *(grades+2)
```

PA1 Q&A

What is a System Call?

- A programmatic way in which a computer program requests a service from the kernel of the operating system it is executed on

- Each system call corresponds to a number defined in a syscalls table.

```
syscall(number, …)
```

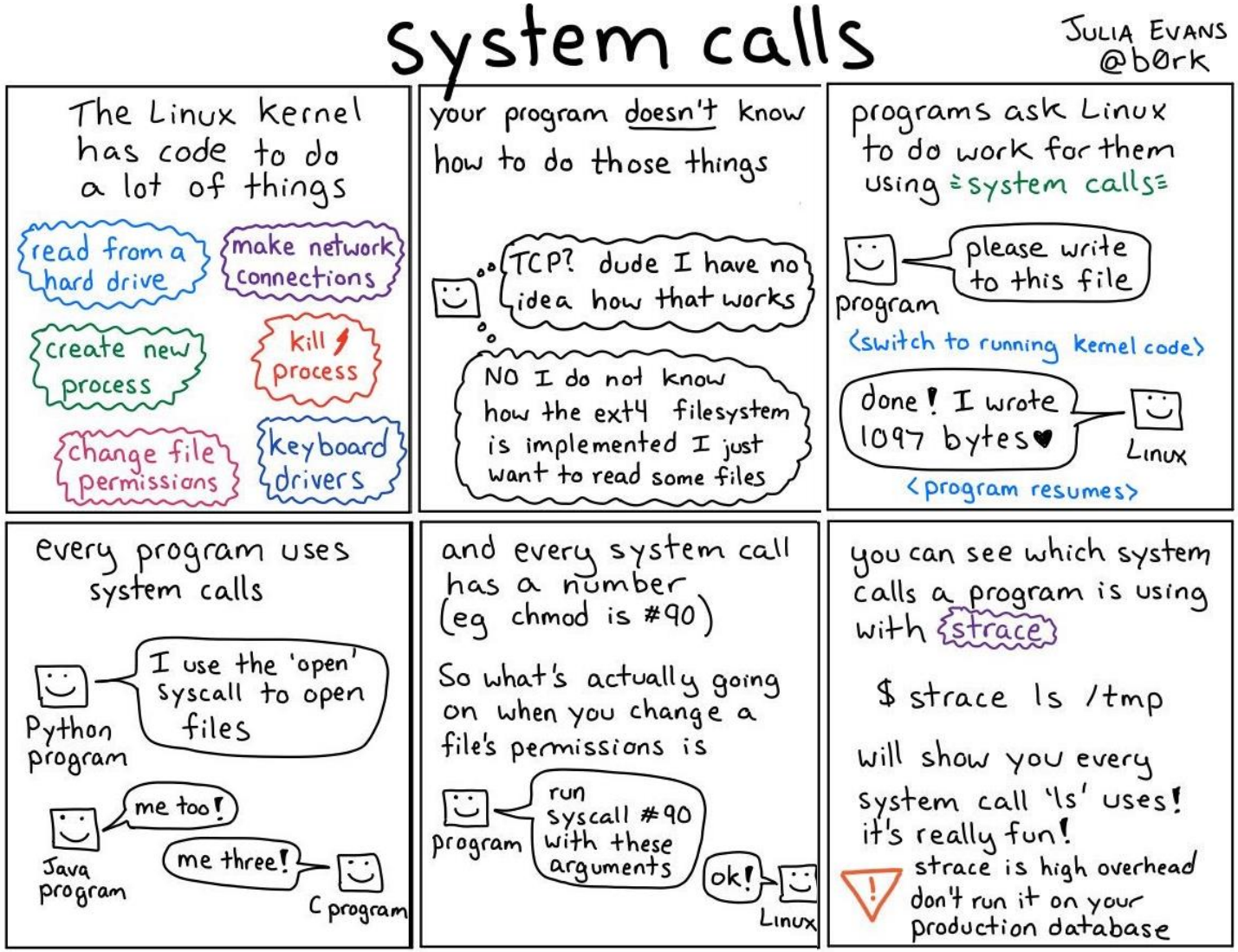
syscall_64.tbl

```
#
# 64-bit system call numbers and entry vectors
#
# The format is:
# <number> <abi> <name> <entry point>
#
# The abi is "common", "64" or "x32" for this file.
#
0     common     read              sys_read
1     common     write             sys_write
2     common     open              sys_open
3     common     close             sys_close
4     common     stat              sys_newstat
5     common     fstat             sys_newfstat
6     common     lstat             sys_newlstat
7     common     poll              sys_poll
8     common     lseek             sys_lseek
9     common     mmap              sys_mmap
10    common     mprotect          sys_mprotect
11    common     munmap                  sys_munmap
```

Do we use system calls at all?

- Example 1:
  - If you open a file using `fopen()` in the library stdio.h, it gets translated into the `open()` system call.
  - In the standard library, the user-space implementation of the `open()` system call executes and passes the system call number.

- Example 2:
  - In Unix-like systems, `fork()` and `exec()` are C-library functions that in turn execute instructions that invoke the `fork()` and `exec()` system calls.



@b0rk on Twitter, or https://drawings.jvns.ca/

Passing Values Between User And Kernel Modes

unsigned long **copy_from_user** (void * to, const void __user * from, unsigned long n);

sizeof()

get_user(x, ptr)

unsigned long **copy_to_user** (void __user * to, const void * from, unsigned long n);

#include <linux/uaccess.h>

put_user(x, ptr)