get_next_line

[kernel session]

/this presentation

- File Handling
- Static Variables

- Practice

/file handling

- 4 main operations
  - What are they?
  - What syscalls can you use?
  - Where are they found in the man?

- Syscalls vs library functions

/file handling

```
int    open(const char *path, int oflag, ...);
```

Return Values

- fd – file descriptor
- -1 in error, sets errno

const char *path

- Path to file from current working directory

int oflag

MUST include one of:

- O_RDONLY
  - open for reading only
- O_WRONLY
  - open for writing only
- O_RDWR
  - open for reading and writing
- O_SEARCH
  - open directory for searching
- O_EXEC
  - open for execute only

Can include other flags!

/file handling - open modes

One mandatory flag from:

- O_RDONLY
    - reading only
- O_WRONLY
    - writing only
- O_RDWR
    - reading and writing
- O_SEARCH
    - open **directory** for searching
- O_EXEC
    - open for execute only

Optional OR (|) Flags:

- O_NONBLOCK
    - do not block on open or for data to become available
- O_APPEND
    - append on each write
- O_CREAT
    - create file if it does not exist
- O_TRUNC
    - truncate size to 0
- O_EXCL
    - error if O_CREAT and the file exists
- And More!!

```
open(const char *path, (O_RDONLY | O_APPEND | O_CREAT);
```

/file handling

```
int     close(int fildes);
```

Return Values

- 0 if successful
- -1 if error
  - file probably already closed/doesn't exist

int fildes (fd)

- fd of previously opened file

leaks!

- Leaving an fd open after the program terminates, is a leak
  - valgrind should catch this, but might need flag

```
$ valgrind --quiet --track-fds=yes ./hello_world
```

/file handling

```
ssize_t    read(int fildes, void *buf, size_t nbyte);
```

Return Values

- Number of bytes read
- -1 in error
  - Sets errno

`int fildes`

- fd in which you're attempting to read from

`void *buf`

- Buffer that will hold the data read, must be large enough to store data

`size_t nbyte`

- How many bytes to read
- BUFFER_SIZE 👀

/file handling

```
ssize_t    write(int fildes, const void *buf, size_t nbytes);
```

Return Values

- nbytes written to fd
- -1 in error
    - Sets errno

`const char *buf`

- What is attempted to write inside of fd

`size_t nbytes`

- How many bytes to write
- Doesn't have to be the same size as *buf 👀

/file handling - library functions

```
FILE    *fopen(const char *path, const char *mode)
int     fclose(FILE *file)
size_t  fread(void *buf, size_t size, size_t nitems, FILE *file)
size_t  fwrite(const void *buf, size_t size, size_t nitems, FILE *file)
```

Not allowed in gnl :(

Handles buffering inside of the functions

Calls the associated sys-call inside of these functions

/questions

What functions will you use in the function get_next_line?

What functions do you need to test your implementation of gnl?

Where to check for errors?

/practice

- Create a text file anywhere on your filesystem that contains a few lines of text using your favorite editor or the command `echo`.

- Let's practice reading from a file. Write a program that: (15 mins)
  - opens that file you made in `read-only` mode,
    - what happens if the file can't be opened? (`chmod -r`)
  - reads the complete contents of the file **using a buffer smaller** than the file content,
  - writes the contents of that buffer onto standard output,
  - closes the file.

/practice

- Now let's practice writing to a file. Write a program that: (10 mins)
  - opens that file you made in `write-only` and `append` mode,
    - what happens if the file can't be opened? (`chmod -w`)
  - writes some additional characters to the file,
  - closes the file.

- Then display the content of your text file in the terminal using your previous program

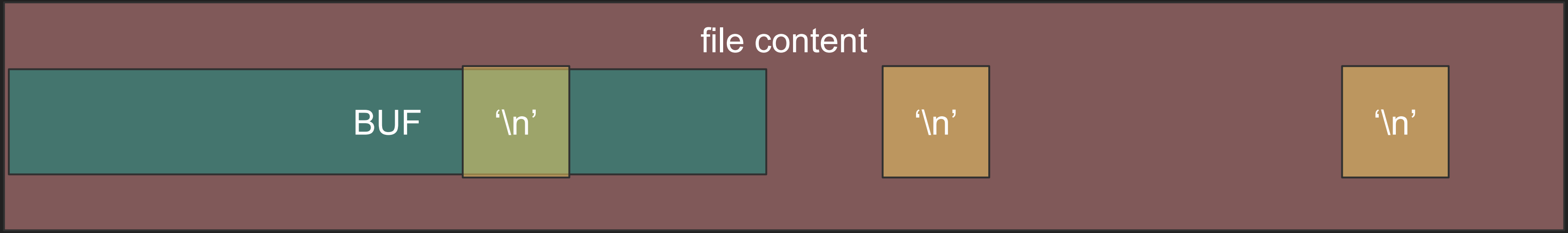
- Check with your neighbor that you did it correctly!

/practice

Make diagrams of possibilities reading a line from a string in various cases. (10 min)

1. Buffer is shorter than file content
2. Buffer is shorter than line
3. Buffer is longer than file
4. No newlines in file

Think about how what needs to be done to print the entire line in each case, then think about what needs to happen for the next line in the file



/static variables

- What is a static variable?
  - Use the internet to find the definition of a static variable and its unique characteristics.

- Discuss the following points together and make sure everyone understands:
  - when might you use a static variable?
  - Where is it allocated in memory?
  - What are the disadvantages when it comes to memory and reusability?
  - When are we allowed to use them in projects?
  - Norm rules regarding static variables

/practice

- Write a function that: (10 mins)
    - declares a `static int`,
    - increments it by 1,
    - then returns the int value.

- Write the accompanying main that calls that function in a loop 9 times and outputs the returned value using `write()` to the standard output on each iteration. What happens to the return value? (10 mins)

- As a closing step, discuss whether it's possible or not to restore a static variable to its initial value.

/practice

- Write a function ft_malloc that: (20 mins)
    - declares a `static int`,
    - initializes that int to 0,
    - increments it by 1,
    - if the int == FT_CHECK it returns NULL
    - otherwise it returns what malloc would return
    - Bonus 1: Print the value of the int, size of the requested pointer and the address returned by this function
    - Bonus 2: give the functions extra parameters and print the name of the file and the line from where it was called
    - Bonus 3: write ft_free that will print the address of the pointer, and the file name and line from where it was called.

- Can you define the value of FT_CHECK during compilation?
- Write the main and replace malloc with ft_malloc in your ft_split, what happens if the 1st malloc fails? What about the 3rd?
- Can you check with valgrind for leaks?