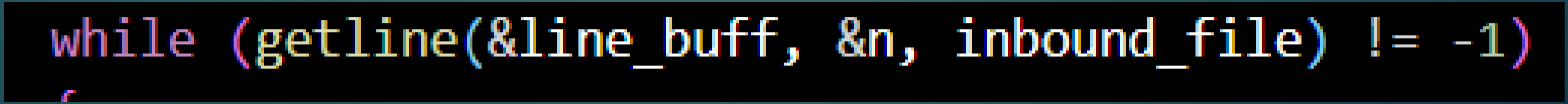
Living sHell

BY PID == 5580

The beginning of the End-of-File

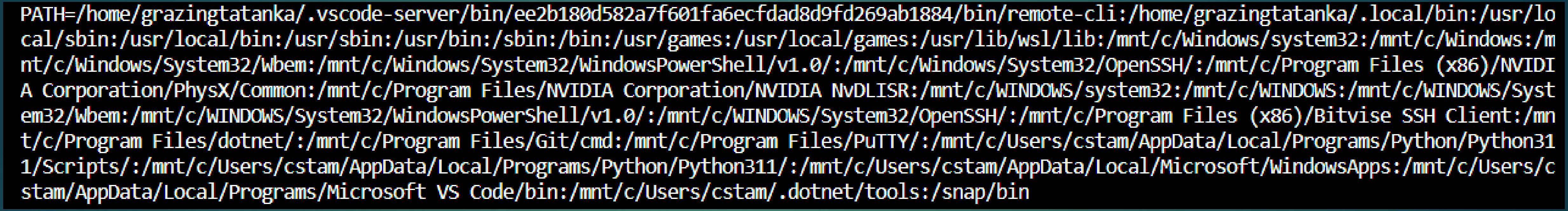
- EOF (End-of-File) condition represented by specific value
- In C this value is (-1); in python it is const. (None)
- <CTRL>+<D> used to indicate EOF in Unix/like systems
- EOF shortcut entered into CLI (Command Line Interface) tells system to send end-of-file character to app or process
- App or process receives this EOF character and performs specific action (typically cleans up and exits)



```
while (getline(&line_buff, &n, inbound_file) != -1)
```

Finding the PATH

- PATH is an environmental variable (use ‘env’ in bash to see)



```
PATH=/home/grazingtatanka/.vscode-server/bin/ee2b180d582a7f601fa6ecfdad8d9fd269ab1884/bin/remote-cli:/home/grazingtatanka/.local/bin:/usr/lo
cal/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/usr/lib/wsl/lib:/mnt/c/Windows/system32:/mnt/c/Windows:/m
nt/c/Windows/System32/Wbem:/mnt/c/Windows/System32/WindowsPowerShell/v1.0/:/mnt/c/Windows/System32/OpenSSH/:/mnt/c/Program Files (x86)/NVIDI
A Corporation/PhysX/Common:/mnt/c/Program Files/NVIDIA Corporation/NVIDIA NvDLISR:/mnt/c/WINDOWS/system32:/mnt/c/WINDOWS:/mnt/c/WINDOWS/Syst
em32/Wbem:/mnt/c/WINDOWS/System32/WindowsPowerShell/v1.0/:/mnt/c/WINDOWS/System32/OpenSSH/:/mnt/c/Program Files (x86)/Bitvise SSH Client:/mn
t/c/Program Files/dotnet/:/mnt/c/Program Files/Git/cmd:/mnt/c/Program Files/PuTTY/:/mnt/c/Users/cstam/AppData/Local/Programs/Python/Python31
1/Scripts/:/mnt/c/Users/cstam/AppData/Local/Programs/Python/Python311/:/mnt/c/Users/cstam/AppData/Local/Microsoft/WindowsApps:/mnt/c/Users/c
stam/AppData/Local/Programs/Microsoft VS Code/bin:/mnt/c/Users/cstam/.dotnet/tools:/snap/bin
```

- Executable files are presumed to be contained in these directories
- **<u>Note that the directories in the PATH are delimited (separated) by colons (:)</u>**
- By default, a shell should check each directory in the PATH for a specified executable…
  - **UNLESS: 1) the specified command is a ‘built-in’ (has special meaning to the shell like cd)**
  - **OR: 2) an absolute file path is specified (absolute paths begin with root dir (/))**

Which main is the main main?

- Your program/application should have an entry point (a main)
- There are multiple prototypes for main:
  - **int main()**
  - **int main(int argc, char *argv[])**
  - **int main(int argc, char **argv)**

You can also specify an additional argument – envp! Like so

int main(int argc, char *argv[], char *envp[])

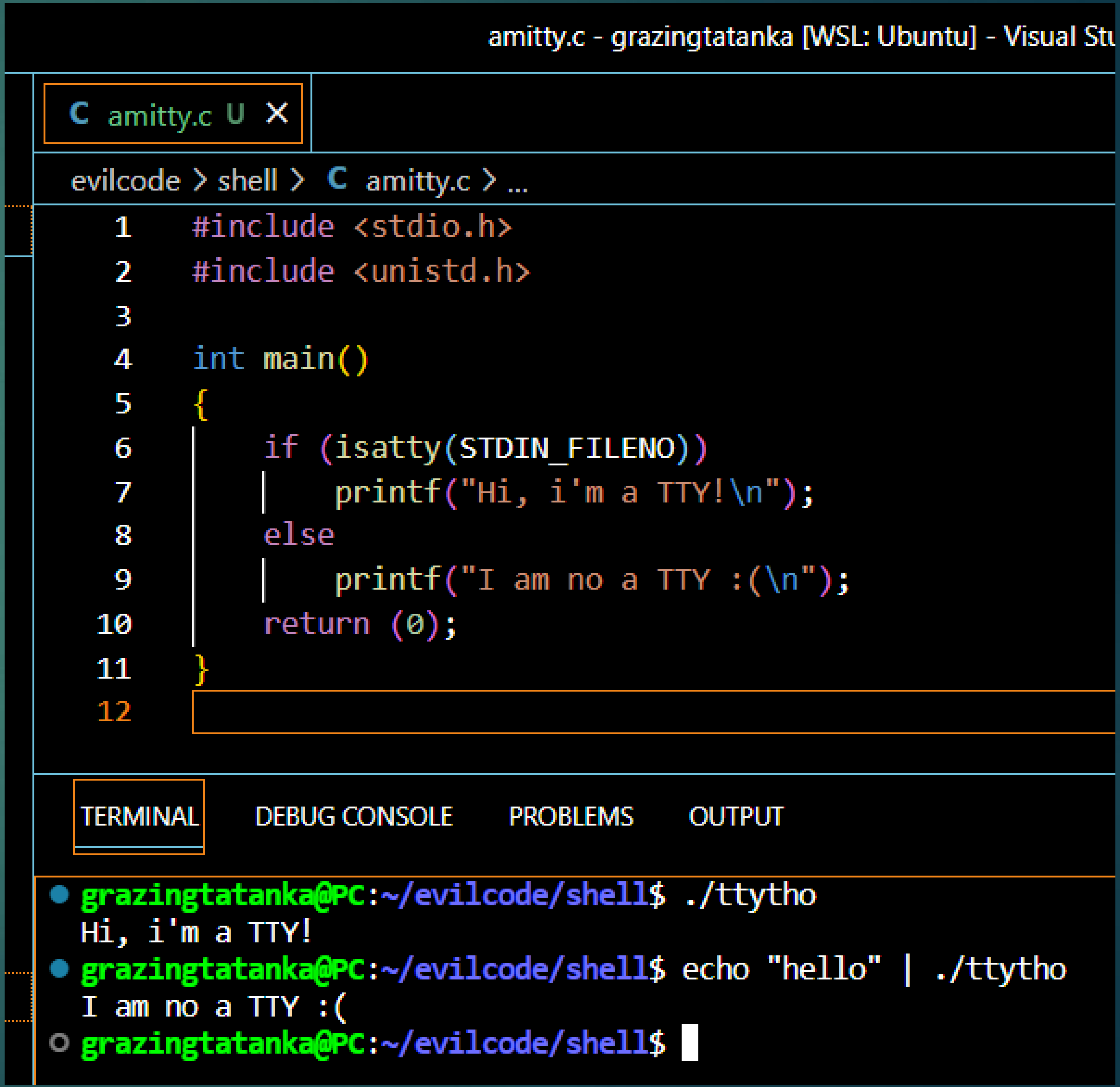
The envp can be thought of as an array of strings, preloaded from environmental variables and in the form of ‘name=value’ (e.g. SHELL=/bin/bash)

That said, it’s often more convenient/practical to simply use getenv()

e.g. `path = getenv("PATH");`

WTF is a TTY?

- TTY is shorthand for a Teletype (used in the before-time)
- Today TTY is used to refer to any terminal device that allows a user to interact with a computer system through a command-line interface (CLI)
- Isatty(FILE_DESCRIPTOR_HERE) in C checks whether the FILE_DESCRIPTOR_HERE is associated with a terminal or not
- Different actions can then be taken depending on the outcome, e.g. printing a prompt message
- This is useful, since it lets us differentiate interactive vs. non-interactive mode use


amitty.c - grazingtatanka [WSL: Ubuntu] - Visual Stu
C amitty.c U ✕
evilcode > shell > C amitty.c > ...
TERMINAL DEBUG CONSOLE PROBLEMS OUTPUT

```c
#include <stdio.h>
#include <unistd.h>

int main()
{
    if (isatty(STDIN_FILENO))
        printf("Hi, i'm a TTY!\n");
    else
        printf("I am no a TTY :(\n");
    return (0);
}

```

```
grazingtatanka@PC:~/evilcode/shell$ ./ttytho
Hi, i'm a TTY!
grazingtatanka@PC:~/evilcode/shell$ echo "hello" | ./ttytho
I am no a TTY :(
grazingtatanka@PC:~/evilcode/shell$ 
```

Function vs. System calls

Function calls:

A function call is a way to execute a block of code that has been defined elsewhere in the program.

The function call transfers control from the current point in the program to the function being called.

The function is executed, and then control returns to the point in the program immediately following the function call.

Function calls are synchronous, meaning that the program waits for the function to complete before continuing.

SYNOPSIS: function calls are used to execute code that has already been defined in the program, while system calls are used to request services from the operating system

System calls:

A system call is a request made by a program to the operating system for a specific service, such as reading or writing to a file or creating a new process.

System calls are made using special functions provided by the operating system, such as open(), read(), write(), fork(), execve(), etc.

The system call transfers control from the program to the operating system kernel, which performs the requested service on behalf of the program.

Once the service is complete, control returns to the program, which continues executing.

System calls are asynchronous, meaning that the program does not wait for the service to complete before continuing. Instead, the operating system notifies the program when the service is complete.

execve won’t let met me be me; instead it wants me to be its ppid

fork() creates a new process
that process is a copy
execve() replaces the current process (copy from fork) with whatever it runs
the parent process waits to reap the bb process

Which PID is me?

PID < 0

<u>Bad fork process</u>

EXITS

EXIT STATUS SHOULD BE NON-0

EXIT ERROR PRINTS TO STDOUT

PID == 0

<u>Child process</u>

EXECUTES

EXEC FAM CALLED HERE

STATUS OF THIS PROCESS EXEC

SHOULD BE RELAYED TO PARENT

REAPED FOLLOWING EXEC

PID > 0

<u>Parent process</u>

WAITS

WAIT FUN FAM USED TO DETERMINE WHEN TO RESUME

RECEIVES EXIT STATUS FROM CHILD PROCESS

REAPS CHILD PROCESS