BEYZA KOMIŞ B231202072
MINIMON PROJECT

IMPORTANT NOTE!

First of inorder to use the linux commnads for resource monitoring I needed a Linux enviromment but because my computer would have trouble with another virtual machine with virtualbox ,
I decided to start my project on windows with VSCode ,afterwards I ran the project on :
 Windows Subsystem For Linux so the commands worked just fine.
Although this is going to be one of the limitations of my project we can at least sucessfully have a working and functional minimon.

1.RECOURCE MONITORİNG

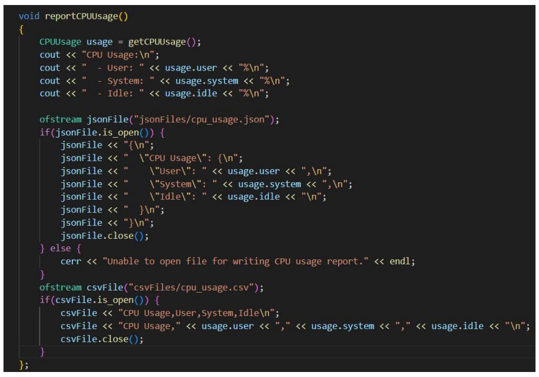
-Functions that handles recource monitıring
- I used structs and vectors for each function to group related data fields and store collections of structs for better code usabilility.

```
vector<ProcessInfo> collectProcesses(int n);
CPUUsage getCPUUsage();
MemoryInfo getMemoryInfo();
DiskInfo getDiskInfo();
NetworkInfo getNetworkInfo();
 vector<ConnectionInfo> getConnectionInfo(int pid);
string getProcessCmdLine(int pid);
string getProcessStatus(int pid);
int getSystemConnectionCount();
 long getProcessRAMUsage(int pid);
  float getProcessCPUUsage(int pid);
```

1. REPORTING

-The below code shows the reporter.h file that every command has its own reporting function
-That prints to console,json and csv files

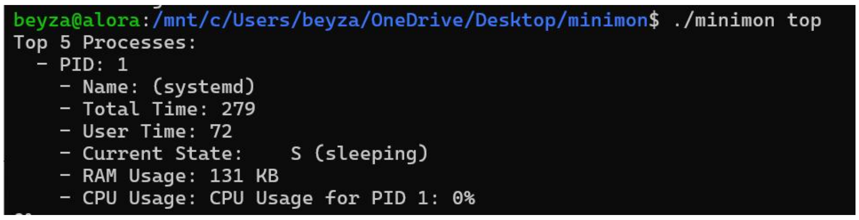
```
void reportCPUUsage();
void reportMemoryInfo();
void reportDiskInfo();
void reportNetworkInfo();
void reportTopProcess(int n);
void reportConnectionInfo(int pid);
```



```
void reportCPUUsage()
{
    CPUUsage usage = getCPUUsage();
    cout << "CPU Usage:\n";
    cout << "  - User: " << usage.user << "%\n";
    cout << "  - System: " << usage.system << "%\n";
    cout << "  - Idle: " << usage.idle << "%\n";

    ofstream jsonFile("jsonFiles/cpu_usage.json");
    if(jsonFile.is_open()) {
        jsonFile << "{\n";
        jsonFile << "  \"CPU Usage\": {\n";
        jsonFile << "    \"User\": " << usage.user << ",\n";
        jsonFile << "    \"System\": " << usage.system << ",\n";
        jsonFile << "    \"Idle\": " << usage.idle << "\n";
        jsonFile << "  }\n";
        jsonFile << "}\n";
        jsonFile.close();
    } else {
        cerr << "Unable to open file for writing CPU usage report." << endl;
    }
    ofstream csvFile("csvFiles/cpu_usage.csv");
    if(csvFile.is_open()) {
        csvFile << "CPU Usage,User,System,Idle\n";
        csvFile << "CPU Usage," << usage.user << "," << usage.system << "," << usage.idle << "\n";
        csvFile.close();
    }
};
```

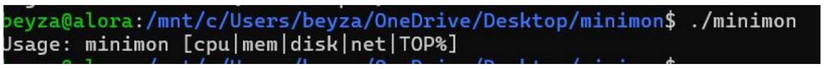
2.USER INTERFACE

-The user inteface is fairly simple and easy to use
- You fisrt have to recomplie with the “make” command
- and run .   ./minimon [cpu|mem|disk|net|TOP%]
- Example :



```
beyza@alora:/mnt/c/Users/beyza/OneDrive/Desktop/minimon$ ./minimon top
Top 5 Processes:
  - PID: 1
    - Name: (systemd)
    - Total Time: 279
    - User Time: 72
    - Current State:    S (sleeping)
    - RAM Usage: 131 KB
    - CPU Usage: CPU Usage for PID 1: 0%
```

- It gives error when you don’t give correct user input:



```
beyza@alora:/mnt/c/Users/beyza/OneDrive/Desktop/minimon$ ./minimon
Usage: minimon [cpu|mem|disk|net|TOP%]
```

THIS IS HOW I HANDLED THE CLI USER INPUT IMPLEMENTATION

```
void userInput(int argc, char *argv[]) {
if (argc < 2)cerr << "Usage: minimon [cpu|mem|disk|net|TOP%]" << endl;
return;
if (strcmp(argv[1], "cpu") == 0) {
   getCPUUsage();
   reportCPUUsage();
} else if (strcmp(argv[1], "mem") == 0) {
   getMemoryInfo();
   reportMemoryInfo();
} else if (strcmp(argv[1], "disk") == 0) {
   getDiskInfo();
   reportDiskInfo();
} else if (strcmp(argv[1], "net") == 0) {
   getNetworkInfo();
   reportNetworkInfo();
}else if (strcmp(argv[1], "top") == 0) {
   int processNum = atoi(argv[2]);
   collectProcesses(processNum);
   reportTopProcess(processNum);
}else if(strcmp(argv[1], "conn") == 0) {
   if (argc < 3)  cerr << "Usage: minimon conn [PID]" << endl; return;
   int pid = atoi(argv[2]);
    reportConnectionInfo(pid);
}else if(strcmp(argv[1], "check") == 0){
      int pid = atoi(argv[2]);
      getProcessCPUUsage(pid);}
else{cerr << "Invalid argument. Use 'cpu', 'mem', 'disk', 'net', or 'top','check'."
<< endl;}}
```

THE PROJECT STRUCUTRE :

```
minimon/
├── README.md
├── Makefile
├── main.cpp        // userInput function that runs first when the project is built
├── minimon            # compiled output (binary)
├── core/            # core logic
│   ├── collector.h
│   ├── collector.cpp
│   ├── analyzer.h
│   ├── analyzer.cpp
│   ├── reporter.h
│   └── reporter.cpp
├── ui/              # user interface (CLI etc.)
│   ├── cli.h
│   └── cli.cpp
└── csvFiles/            // report csv files located here
    ├── cpu_usage.csv
    ├── disk_info.csv
    ├── memory_info.csv
   └── network_info.csv
   └── top_process.csv
└── jsonFiles/            // report json files located here
    ├── cpu_usage.json
    ├── disk_info.json
    ├── memory_info.json
   └── network_info.json
   └── top_process.json
```

THE MAKEFILE

- The purpose of this Makefile is to automate the process of compiling and building your C++ project (minimon).

```
CXX = g++
CXXFLAGS = -Wall -std=c++17 -Icore -Iui
TARGET = minimon
SRC = main.cpp \
     core/collector.cpp \
     core/analyzer.cpp \
     core/reporter.cpp \
     ui/cli.cpp \
OBJ = $(SRC:.cpp=.o)
all: $(TARGET)
$(TARGET): $(OBJ)
   $(CXX) $(CXXFLAGS) $(OBJ) -o $(TARGET)
%.o: %.cpp
   $(CXX) $(CXXFLAGS) -c $< -o $@
clean:
   rm -f $(OBJ) $(TARGET)
```

FILTERING PARTS OF THE PROJECT

-The filtering is mostly done in the functions of the monitoring resource and reporting layer .

Some example of filtering I Have Done:

Memory Info:
-Specifically check for "MemTotal:", "MemFree:", and "MemAvailable:" from /proc/meminfo.
(This is selective extraction — a basic form of filtering.)

Disk Info:
-Filtering lines starting with "sda" from /proc/diskstats.

Network Info:
Checking if the device name is "eth0", "wlan0", or "enp0s3" from /proc/net/dev.
(Again, this is filtering based on known device names.)

-Although I implemented a few functions for the analyzer module, they didn’t play a significant role in this project due to the limited scope and the relatively small set of processes involved.

Functions like :

```
CPUUsage calcultePrevandNewUsage(const CPUUsage& oldUsage,const CPUUsage& newUsage);
bool isCPUUsageIncreased(const CPUUsage& usage, float threshold = 80.0f);
vector<ProcessInfo> sortProcess(const vector<ProcessInfo>& processes);
```

 were included with the intention of performing trend analysis and prioritization, but in practice, they weren’t particularly useful given the simplicity of the system data being handled.

USED THE NECESSARY COMMANDS TO READ FROM THE LINUX FILES

-Reading from Linux virtual files that act as system interfaces:

/proc/[pid]/stat (process stats)
/proc/[pid]/status (process status)
/proc/[pid]/cmdline (process command line)
/proc/meminfo (memory and swap info)
/proc/stat (CPU stats)
/proc/diskstats (disk I/O stats)
/proc/net/dev (network interface stats)
/proc/net/tcp, /proc/net/udp, /proc/net/tcp6 (network connections)

SOME LINUX LIBRARIES I USED :

Linux/System Libraries

<dirent.h>: For directory operations (opendir, readdir, closedir) to iterate over /proc entries.
<arpa/inet.h>: For network address conversion (inet_ntoa, htonl), used in IP/port parsing.
<unistd.h>: For POSIX functions like usleep, sysconf.
<sys/statvfs.h>: For filesystem statistics (statvfs), used to get disk space info.

THE LOGIC BEHIND MY RESOURCE MONITORING FUNCTIONS.

Here is an example function:

```
vector<ProcessInfo> collectProcesses(int n) {
   vector<ProcessInfo> processes;
   DIR* dir = opendir("/proc");
   if (!dir) cout << "Could not open /proc directory." << endl; return processes;}
   struct dirent* entry;
   while ((entry = readdir(dir)) != nullptr) {
      if (!isdigit(entry->d_name[0])) continue; // Only numeric dirs (PIDs)
      string pidStr = entry->d_name;
      string statPath = "/proc/" + pidStr + "/stat";
      ifstream statFile(statPath);
      string line;
      if (!statFile.is_open()) continue;
      int pid;
      string comm;
      char state;
      long dummy, utime, stime;
      statFile >> pid >> comm >> state;
      // Skip to utime (14th) and stime (15th) fields
      for (int i = 0; i < 11; ++i) statFile >> dummy;
      statFile >> utime >> stime;
      ProcessInfo p;
      p.pid = pid;
      p.name = comm;
      p.utime = utime;
      p.stime = stime;
      p.status= getProcessStatus(pid);
      p.ramUsageGB = getProcessRAMUsage(pid);
      processes.push_back(p);}
   closedir(dir);
if (n > 0 && static_cast<size_t>(n) < processes.size()) {
      processes.resize(n); }
        return processes;
}
```

LOGIC BEHIND THIS FUNCTION AND THE REST OF THE FUNCTIONS

Logic of collectProcesses(int n)

1.Open the /proc directory:
This directory contains a subdirectory for each running process, named by its PID (process ID).

2.Iterate over directory entries:
For each entry in /proc, check if the name is numeric (i.e., it's a process directory).

3.For each process:
-Build the path to /proc/[pid]/stat, which contains process statistics.
-Open and read this file to extract process information:
PID, command name, state, user time (utime), system time (stime), etc.
-Create a ProcessInfo struct and fill it with the parsed data.
-Get additional info by calling helper functions:
getProcessStatus(pid) for the process state.
getProcessRAMUsage(pid) for memory usage.
-Add the struct to a vector of processes.
4.Sort the vector by total CPU time (descending), so the most active processes are first.
5.Resize the vector to keep only the top n processes, if n is specified.
6.Return the vector of ProcessInfo structs.

How This Pattern Appears in Other Resource Monitoring Functions

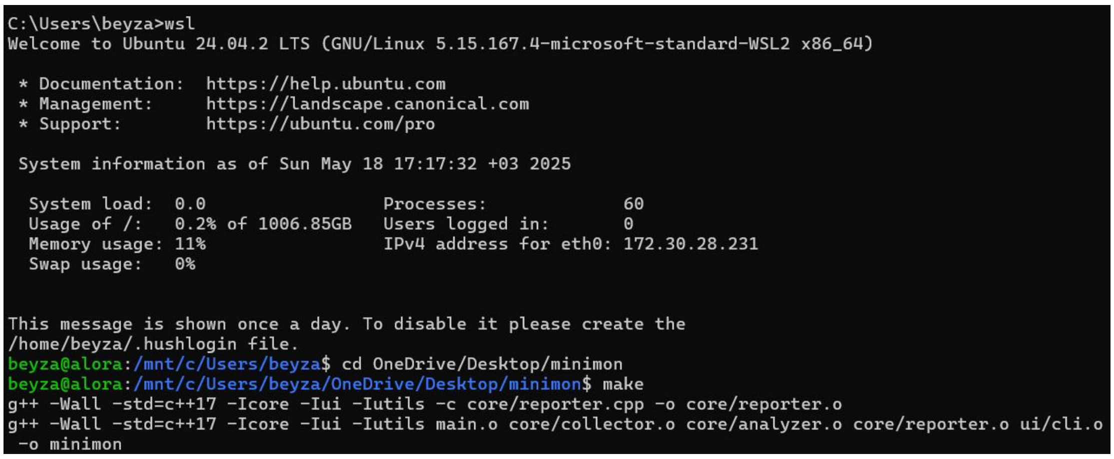
I use a very similar pattern for all your resource monitoring:

-Open a special file in /proc (or /proc/net, /proc/meminfo, etc.).
-Iterate over lines or entries in that file.(WHICH is a form of filetring thourgh necessary info)
-Parse each line (using istringstream and string manipulation) to extract the relevant data.
-Store the data in a struct (like MemoryInfo, DiskInfo, NetworkInfo, etc.).
-Return the struct (or a vector of structs) for reporting or further processing.
Examples:
Memory: - Open /proc/meminfo, parse lines for MemTotal, MemFree, etc., fill a MemoryInfo struct.
Disk: Open /proc/diskstats, parse lines for device stats, fill a DiskInfo struct.
Network: Open /proc/net/dev, parse lines for interface stats, fill a NetworkInfo struct.
Connections: - Open /proc/net/tcp or /proc/[pid]/net/tcp, parse lines for connection info, fill a ConnectionInfo struct.
In short: -I am using a repeatable, modular pattern:
Open → Iterate → Parse → Store in struct → Return
This makes your code organized, reusable, and easy to extend for new resource types!

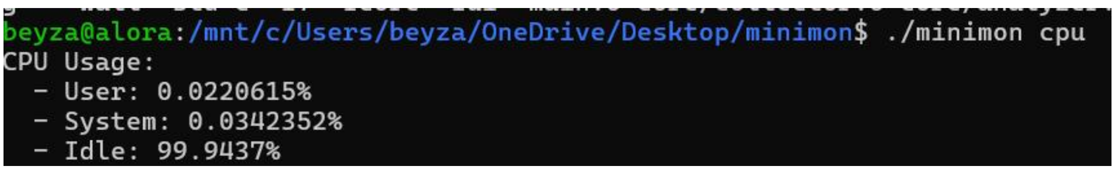
HOW TO RUN THE APPLICATION

1. FIRST ACCESS YOUR LINUX ENVIROMENT
2. GO TO THE PROJECT DIRECTTORY
3. COMPİLE WİTH THE MAKE COMMAND
4. AND RUN THE COMMANDS BELOW

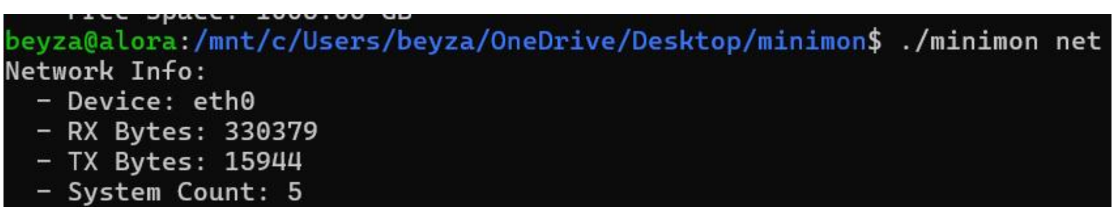
SCREENSHOTS OF MY APPLICATION



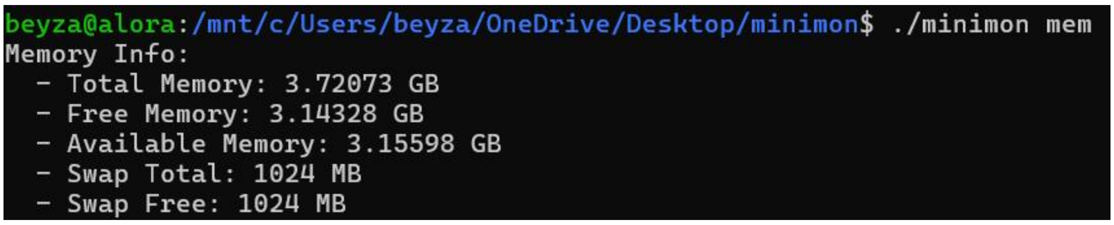
```
C:\Users\beyza>wsl
Welcome to Ubuntu 24.04.2 LTS (GNU/Linux 5.15.167.4-microsoft-standard-WSL2 x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/pro

 System information as of Sun May 18 17:17:32 +03 2025

  System load:  0.0                Processes:             60
  Usage of /:   0.2% of 1006.85GB  Users logged in:       0
  Memory usage: 11%                IPv4 address for eth0: 172.30.28.231
  Swap usage:   0%


This message is shown once a day. To disable it please create the
/home/beyza/.hushlogin file.
beyza@alora:/mnt/c/Users/beyza$ cd OneDrive/Desktop/minimon
beyza@alora:/mnt/c/Users/beyza/OneDrive/Desktop/minimon$ make
g++ -Wall -std=c++17 -Icore -Iui -Iutils -c core/reporter.cpp -o core/reporter.o
g++ -Wall -std=c++17 -Icore -Iui -Iutils main.o core/collector.o core/analyzer.o core/reporter.o ui/cli.o
 -o minimon
```



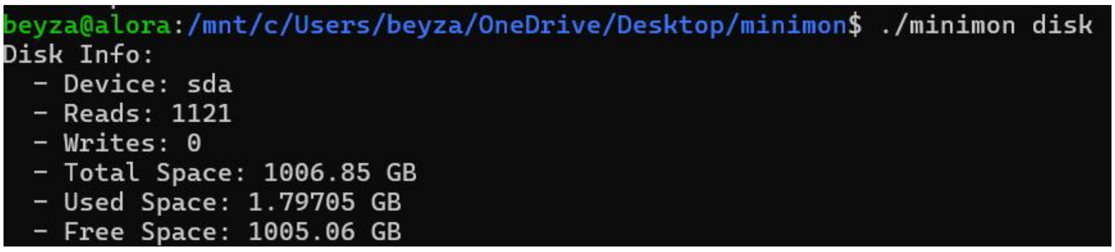
```
beyza@alora:/mnt/c/Users/beyza/OneDrive/Desktop/minimon$ ./minimon cpu
CPU Usage:
  - User: 0.0220615%
  - System: 0.0342352%
  - Idle: 99.9437%
```



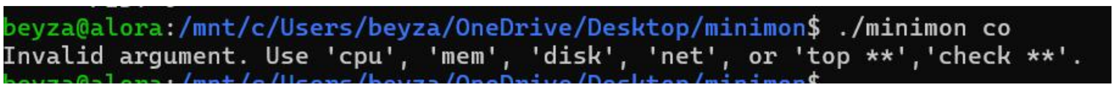
```
beyza@alora:/mnt/c/Users/beyza/OneDrive/Desktop/minimon$ ./minimon net
Network Info:
  - Device: eth0
  - RX Bytes: 330379
  - TX Bytes: 15944
  - System Count: 5
```



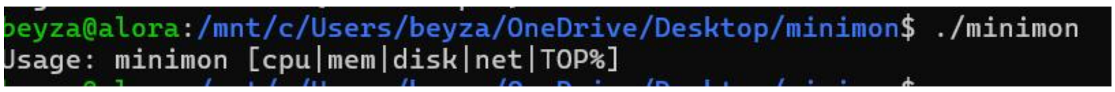
```
beyza@alora:/mnt/c/Users/beyza/OneDrive/Desktop/minimon$ ./minimon mem
Memory Info:
  - Total Memory: 3.72073 GB
  - Free Memory: 3.14328 GB
  - Available Memory: 3.15598 GB
  - Swap Total: 1024 MB
  - Swap Free: 1024 MB
```



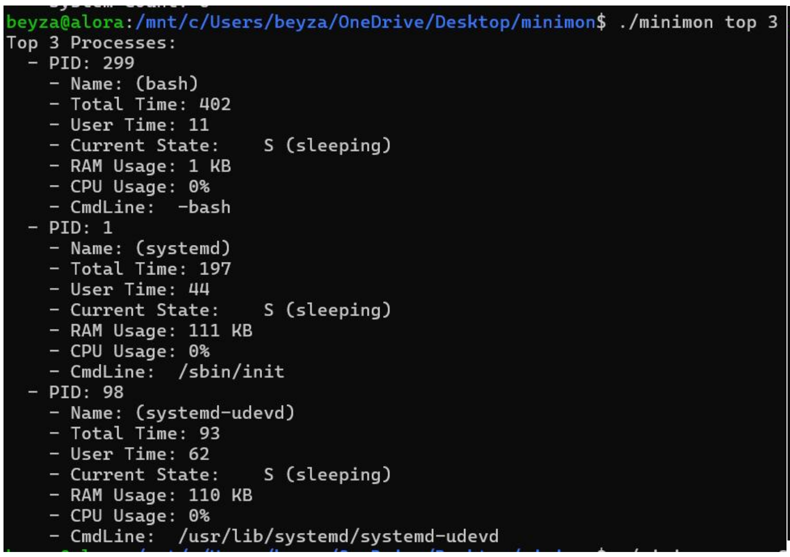
```
beyza@alora:/mnt/c/Users/beyza/OneDrive/Desktop/minimon$ ./minimon disk
Disk Info:
  - Device: sda
  - Reads: 1121
  - Writes: 0
  - Total Space: 1006.85 GB
  - Used Space: 1.79705 GB
  - Free Space: 1005.06 GB
```



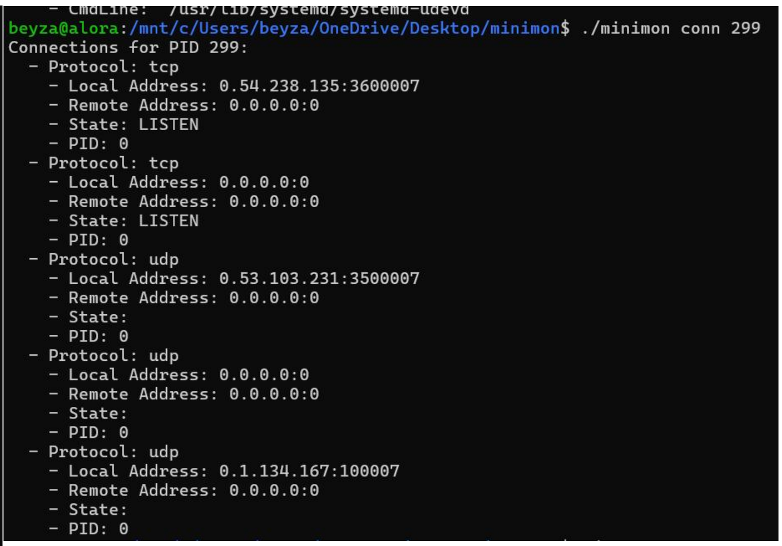
```
beyza@alora:/mnt/c/Users/beyza/OneDrive/Desktop/minimon$ ./minimon co
Invalid argument. Use 'cpu', 'mem', 'disk', 'net', or 'top **','check **'.
```



```
beyza@alora:/mnt/c/Users/beyza/OneDrive/Desktop/minimon$ ./minimon
Usage: minimon [cpu|mem|disk|net|TOP%]
```



```
beyza@alora:/mnt/c/Users/beyza/OneDrive/Desktop/minimon$ ./minimon top 3
Top 3 Processes:
  - PID: 299
    - Name: (bash)
    - Total Time: 402
    - User Time: 11
    - Current State:    S (sleeping)
    - RAM Usage: 1 KB
    - CPU Usage: 0%
    - CmdLine:  -bash
  - PID: 1
    - Name: (systemd)
    - Total Time: 197
    - User Time: 44
    - Current State:    S (sleeping)
    - RAM Usage: 111 KB
    - CPU Usage: 0%
    - CmdLine:  /sbin/init
  - PID: 98
    - Name: (systemd-udevd)
    - Total Time: 93
    - User Time: 62
    - Current State:    S (sleeping)
    - RAM Usage: 110 KB
    - CPU Usage: 0%
    - CmdLine:  /usr/lib/systemd/systemd-udevd
```



```
    - CmdLine:  /usr/lib/systemd/systemd-udevd
beyza@alora:/mnt/c/Users/beyza/OneDrive/Desktop/minimon$ ./minimon conn 299
Connections for PID 299:
  - Protocol: tcp
    - Local Address: 0.54.238.135:3600007
    - Remote Address: 0.0.0.0:0
    - State: LISTEN
    - PID: 0
  - Protocol: tcp
    - Local Address: 0.0.0.0:0
    - Remote Address: 0.0.0.0:0
    - State: LISTEN
    - PID: 0
  - Protocol: udp
    - Local Address: 0.53.103.231:3500007
    - Remote Address: 0.0.0.0:0
    - State:
    - PID: 0
  - Protocol: udp
    - Local Address: 0.0.0.0:0
    - Remote Address: 0.0.0.0:0
    - State:
    - PID: 0
  - Protocol: udp
    - Local Address: 0.1.134.167:100007
    - Remote Address: 0.0.0.0:0
    - State:
    - PID: 0
```

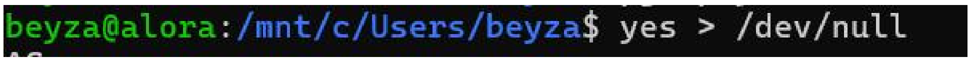
NOTE :

- Do you notice how the cpu usage is zero for every process ,that is because all of my processes are sleeping but inorder to check if my is function below is working , I had a little experinment.

Float getProcessCPUUsage(int pid)

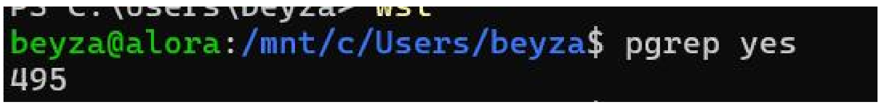
- I ran an infinite loop in another terminal
- Found its pid
- Checked that cpuusage utilization with this function
- Found out that it was working perfectly …

Runs forever



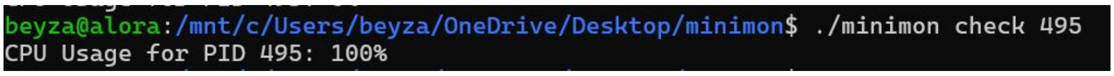
```
beyza@alora:/mnt/c/Users/beyza$ yes > /dev/null
```

Check its pid



```
beyza@alora:/mnt/c/Users/beyza$ pgrep yes
495
```

And use it with your function



```
beyza@alora:/mnt/c/Users/beyza/OneDrive/Desktop/minimon$ ./minimon check 495
CPU Usage for PID 495: 100%
```