README

After Downloading, Move A1_FINAL.c to the home dir and compile there (untested o/w)

How to compile code:

```
 gcc -o MySystemStats A1_FINAL.c -lm
```

(-o MySystemStats optional)

Executable: ./MySystemStats

→ with commands from Commands section after (ie. ./MySystemStats --system)

Commands:

--system: prints system sections (memory and CPU Cores and CPU usage)
--user: prints user section
--graphics: prints bonus graphical output for the memory and CPU usage
--samples=N will change the program to take N samples
--tdelay=tdelay will change the program to pause between each sample for tdelay seconds
Positional arguments: if two **positive** integers are imputed with a space in between, the program will take the first to be the # of samples to be collected, and the second to the time delay between each sample

- If both --system and --user are called, the program will print both system and user sections
- If neither is called, the program will print both
- The default number of # of sampleis 10, and default time delay is 1s
- All invalid CLAs will terminate the program and print an error message with the CLA that caused the issue
  - ie. non-integer or negative positional args or non-integer or negative N/tdelay values, or misspelled --command flags
- Similarly all errors checked for in the code will terminate the program and print an error message indicating what went wrong.

Structs Defined for the Program

MemData has:

- double parameters physUsed, physTotal, virtUsed, virtTotal
- This stores the memory (physical used, physical total, virtual used, virtual total) in GB

CPUData has:

- long array parameter cpuuse of length 10
  - Stores all time “time spent” doing tasks or idled from /proc/stat at a snapshot
    - https://www.idnt.net/en-US/kb/941772
- long parameter sum
  - Stores sum of all time spent doing tasks
- int parameter error
  - Stores whether or not there was an error in reading the file /proc/stat, 1 in error, 0 o/w
- Double parameter usage
  - Stores CPU usage in % at the second snapshot
  - In array of CPUData cpudata, cpudata[i] has a usage of 0.

---Functions---

Three types of functions

- ones that preform more than one function (//PRINT AND INPUT INFORMATION)
    - read information (from file/libraries) and store info
    - prints information
    - preforms some kind of calculation
- Parses Command Line Arguments (processes user inputd information)
- Helper Functions that only preform a specific calulation

Below is all the Functions used in the program (excluding main)

```
//PARSE COMMAND LINE ARGUMENTS
int parseCLA(int argc, char **argv, int toprint[4],int *Nptr, int *tdelayptr);

//PRINT AND INPUT INFORMATION
int printRunningParam(int N, int tdelay); //input info and print
int updateMemory(MemData *memdata, int i); //take samples updates
int printMemory(MemData* memdata, int i, int N, int seq,int graphics);
//calculate & print
int printUsers(); //take samples and print
int printCores(); //input info and print
void printCPUUsage(CPUData prev, CPUData curr,CPUData *cpudata,int i,int
graphics); //calculate and print
int printSysInfo(); // input info and print
CPUData updateCPU(); //input info

//helper functions
void copyCPUData(CPUData *dest, CPUData *src);
double convertbytes(long bytes, int unit); //convert amount bytes in unit
unit(where unit = #of bytes per one unit of bytes) to GB
int parseInt(char *line);
void convertSec(long sec, int time[4]);
```

Additional details about each function are written as comments in A1_FINAL.c

How Samples Are Taken & Screen Refreshing is Done:

- All pausing of the program to take samples occur in main Function

-memdata → an array of type MemData of length n that stores all samples of memory data

-cpudata→ an array of type CPUData of length n+1 that stores all snapshots of /proc/stat and usage.

1st Iteration

1) For CPU usage we first take a snapshot of /proc/stat and store it in CPUData curr using updateCPU(); and the copy it to CPUData prev and in the CPU data array that stores all snapshots, memdata at index **0** (using copyCPUData)
2) It will then print everything else:
    - running parameters
    - first sample of the memory usage by using printMemory() and passing through
    - users (if indicated by our CLAs)
    - #of cores
3) Pause the program for tdelay seconds
4) Then take a second snapshot and store it in current (using updateCPU), then calculate the usage and print it, and store that usage value to the in cpudata at index **1** in in printCPUUsage().

Looping

We start looping at 1 because we've already printed the first iteration, and pause at the beginning of every iteration.

5) The whole screen is cleared and it reprints all the running parameters and all samples of memory(previous), and takes a sample of the current memory usage, current connected users and # of cores, reprints the previous CPUUsage
    a) If the program is indicated to run sequentially
        i) it will not clear screen
        ii) And print a blank line where the previous samples of Memory are

6) Pauses the program
7) then takes the next snapshot of CPU usage -- storing it in cpudata at index **i+1,** and prints it.

Once the Loop is Over is over, it will print the System information:

- system name
- machine name
- OS version
- OS release
- machine architecture
- time that the system has been running since its last reboot

The Following Section will Tackle How Each Section (Memory, User, CPU core and Usage and System Information is calculated)

---Memory---

All memory is read from the library <sys/sysinfo.h>, and found in parameters of the struct of type sysinfo named sysinfoData which is populated when sysinfo(&sysinfoData) is called.

1. Used Physical Memory ->totalram-freeram
2. Total Physical Memory->totalram
3. Total Virtual Memory -> totalram+totalswap
4. Used Virtual Memory ->total virtual memory - freeram- freeswap

Stored in GB and converted using covertbytes()

Accessing and storing the information is done by updateMemory

--User--

All Users are read “line by line” in printUser() from <utmp.h>

Printed User information

1. Username in ut_user
2. tty in ut_line
3. host -- host will indicate which session number a user logging in from ssh in the form ‘%0.sessionNumber’

--CPU Cores--

Taken from <unistd.h> using function sysconf(_SC_NPROCESSORS_CONF) which will return number of cores.

--CPU Usage--

https://www.idnt.net/en-US/kb/941772 From this website, I understood %CPU Usage to be the change in idle time over the change in the total time spent

This can only be taken if there is a previous snapshot to compare it with so that was why I took the samples the way I did.

In my program this is calculated in printCPUUsage(), and this snapshot is stored in cpudata in this function as well

→ Originally, I left the first iteration of CPU usage blank, and only started printing usage on the 2nd iteration, but that misses 1 sample taken for the CPU usage.

--System Information --

System info ->

- system name
- machine name
- OS version
- OS release
- machine architecture

Is taken from the <sys/utsname.h> library

- time that the system has been running since its last reboot is taken from the <sys/sysinfo.h> library under parameter uptime.

Graphics

Please Full Screen the Command Terminal when using graphics.

1. Memory Usage adds a “block”, the desired printed char to represent increase/decrease, for every 0.01 GB increased/decreased from the **LAST MEMORY SAMPLE OF PHYS.USED**
    a. The first graphic display is always ‘|o’ (positive and 0 block difference) since there is no earlier sample to compare it to
2. CPU Usage adds a bar for every 0.2% of CPU Usage.
    a. I did this because I believe it more clearly displays the change/increase in CPU Usage because you can compare all samples to the first CPU Usage
    b. This will typically work better when CPU usage is low, when, ie. % usage is below 32 %, which is okay within the boundary of normal CPU usage (ie. web browsing and microsoft office)
        i. https://www.avg.com/en/signal/fix-high-cpu-usage#:~:text=When%20your%20computer%20is%20idle,the%20latest%20GTA%205%20mods.