B09 Assignment 1- System Monitoring Tool

This is a system monitoring tool written in C

*******WARNING: PUT YOUR TERMINAL ON FULL SIZE BEFORE USE**

How to Compile (default)

1. In the directory with all the other `.c` files (you can check with `ls`), type in `gcc main.c`
2. run `./a.out` to execute the code

Usage Rules

Here is a list of different usages for this program

```
#default, will do 10 samples with 1 second refresh
./a.out

#Only the system usage should be generated
./a.out --system

#Only the users usage should be generated
./a.out --user

#output will be sequential without refresh
./a.out --sequential

#if you decide to output to a file, GIVE IT SOME TIME its a bit slow :(
./a.out --sequential > bruh.txt

#you can also put positional arguements, but there must be 2
#5 samples, 2 second delay
./a.out 5 2

#It is also fine if you chain it with other flags before or after
./a.out --system 5 2
./a.out 5 2 --system

#can be done sequentially
./a.out 5 2 --sequential

#There must be always 2 positive integers. The following are unvalid:
./a.out 5
./a.out 5 --system
./a.out --system 3
./a.out --sequential 5
./a.out -5 2
./a.out 5 -2

#notice there is only one number. if you want to execute with just numbers, you need to specify BOTH of them as stated in the assignment


#if you only want to change the sample OR just change the delay you can include the --samples=N or --user

#just 5 samples, tdelay is still 1. NO NEGATIVE NUMBERS ARE ACCEPTED
./a.out --samples=5

#just 4 second delay, samples is still 10 NO NEGATIVE NUMBERS ARE ACCEPTED
./a.out --tdelay=4

#5 second delay, 5 samples
./a.out --samples=5 --tdelay=5


#Some complex examples

#if we have both the --samples/--tdelay flags enabled, the code will still run with positional arguements and it will process on whichever

#3 samples and 3 second delay since it is the most recent update
```

```
./a.out -samples=2 -tdelay=2 3 3

#2 samples and 2 second delay since it is the most recent update
./a.out 3 3 -samples=2 -tdelay=2
```

commonLibs.h

This code includes various system headers such as `<sys/resource.h>`
, `<sys/utsname.h>`
, `<sys/types.h>`
, `<sys/sysinfo.h>`
, `<utmp.h>`
, `<unistd.h>`
, `<stdio.h>`
, `<stdlib.h>`
, `<string.h>`
, `<ctype.h>`
, `<time.h>`
, `<sys/times.h>`
, `<sys/time.h>`
, `<sys/stat.h>`
, and `<sys/types.h>`

These headers provide various system-level functions and variables for the program.

main.c

The code has two functions: `seqFlag` and `refresh` that display the information on the terminal. The information includes system memory usage, number of sessions, number of cores, machine information and CPU usage. The code includes five external libraries `memUsage.c, sessionUsers.c, cores.c, machineInfo.c, parseArg.c` to display the required information. The `parseArgs` function in the `parseArg.c` library parses the command-line arguments to set the default values of the information to be displayed and the frequency of the display. The `clear_screen` function is used to clear the terminal with ANSI escape codes before printing the updated information.

machineInfo.c

This is a C program that uses the `uname` function to retrieve information about the system and print it to the standard output. The program includes a header file called `commonLibs.h`, and defines a function called `printMachineInfo` which retrieves and displays information about the system using the `uname` function. The information displayed includes the system name, node name, release, version, and machine. The information is printed in a formatted manner with descriptive labels for each piece of information.

cores.c

This code implements the functions `logCores`, `logCpuUsage` and `get_stats` to print the number of cores on the system and the CPU usage. `logCores` prints the number of cores on the system using the `sysconf` function. `logCpuUsage` calculates the CPU usage by reading values from the `/proc/stat` file and computing the percentage of CPU utilization. It does this by first reading the file once, sleeping for a second, then reading it again and computing the difference in the CPU utilization between the two readings. `get_stats` reads the `/proc/stat` file and stores the values in a struct `cpustat`. `calculate_load` computes the CPU utilization from two `cpustat` structs.

memUsage.c

This code prints the memory usage information for the system and for the current process. It does so by first calling the `sysinfo` function to get the system information and the `getrusage` function to get the resource usage statistics for the current process. It prints the physical memory usage (total and used) in gigabytes and the virtual memory usage (total and used) in gigabytes. It repeats the process for `NUM_SAMPLES` times with an interval of `SLEEP_TIME` seconds between each iteration.

parseArg.c

This code parses the command line arguments passed to a C program. It uses the `argc` and `argv` arguments from `main()` and updates the values of several variables, such as `systemm`, `user`, `sequential`, `samples`, and `tdelay`, based on the arguments specified by the user. The code supports both positional arguments and named options with the format `--flag=value`.

The code checks for the presence of flags such as `--system` and `--user`, and updates the values of the corresponding variables. For named options such as `--samples=` and `--tdelay=`, the code extracts the integer value following the equal sign and updates the `samples` and `tdelay` variables.

The code also performs several checks to ensure that the arguments passed by the user are valid, such as checking that the values for `samples` and `tdelay` are positive integers and that the correct number of positional arguments are provided. If an invalid argument is found, the code will print an error message and exit the program with a status code of 1.

sessionUsers.c

This code uses the `utmp` file and the `getutent()` function to print information about users and sessions. The utmp file stores information about the users and sessions, and the `getutent()` function retrieves this information. The program starts by opening the utmp file using the `setutent()` function and then calling the `logSessional()` function to print the information about the users and sessions.

The `logSessional()` function uses a loop to retrieve each record from the utmp file using the `getutent()` function and prints the information about the user and session if the record's type is USER_PROCESS. After printing the information, the program closes the utmp file using the `endutent()` function.