Using pipe and fork in stats_main.c

1. The goal is to have one process launched per function reporting memory utilization, connected users and CPU utilization, and using pipe to communicate among the parent processor and its child processors. This is implemented in my stats_main.c, int main(int argc, char **argv)
2. I introduce three pipefd[2], they are pipeMem[2](used to let the processor write memory utilization to parent process), pipeUse[2](used to let the processor write connected user to parent process), pipeCpu[2](used to let the processor write CPU utilization to parent process)
3. In my main, there is a Loop to iterate through the sample size. At that while loop, I use fork three times to create three processors to implement per functions reporting the memory utilization, connected users and CPU utilization, use `write()` to write to corresponding pipe, then let child processor exit. In the parent processor, the parent processor use `read` to gather three sections information and print them out in correct structure. And also during the while loop, I have close the unused pipe end, close the read end in each child processor, close write end after `write`, close write end in parent processor, close `read` end after `read()`.

Command line arguments

This program is written in C under Linux.
And it accepts several **command line arguments**:

- **--system** : to indicate that only the system usage should be generated
- **--user**: to indicate that only the users usage should be generated
- **--graphics**: to include graphical output in the cases where a graphical outcome is possible as indicated below. **-g** is same as **--graphics**
- **--sequential**: to indicate that the information will be output sequentially without needing to "refresh" the screen
- **--sample=N**: if used the value *N* will indicate how many times the statistics are going to be collected and results will be average and reported based on the N number of repetitions. **If not value is indicated the default value will be 10**
- **tdelay=T** : to indicate how frequently to sample in seconds. **If not value is indicated the default value will be 1**
- There is a short way to indicate the sample size and tdelay time, input two positive integer in this order (**samples tdelay**) ex) **8 2** where 8 is the sample size and 2 is the tdelay time in seconds

- Ctrl + z, ignore
- Ctrl + c, it will ask you to quit the program or not, if input 'y' or 'Y', then quit program, if input 'n' or 'N', then continue the program, if press other letter, will also quit the program

Reproted "Stats" and How to fetch them

- **running parameters**:
  - **number of samples** (default value is 10, can be set by command line **--sample=N**)
  - **samples frequency** (default value is 1, can be set by command line **--tdelay=T**)
    - samples and tdelay can also be set by command line with two positive integer **sample tdelay**
  - **memory self-utilization** : To get the total memory usage in kilobytes, I used the `ru_maxrss` field from the `struct rusage` defined in **<sys/resource.h>**. In order to fetch the information, `getrusage()` function is required.
- **user usage**:
  - **report how many users are connected in a given time**
  - **report how many sessions each user is connected to**
    - To get **user usage** information, I used `struct utmp` defined in **<utmp.h>**. In order to fetch the information, `getutent()` function is required. Following key words are useful.
      - `ut_user`: the user log-in name
      - `ut_line`: the device name
      - `host`: the host name
- **system usage**:
  - I use `printf("\x1b[s");` to save the cursor position, use `printf("\x1b[u");` to go back to saved position, which implemented the rewrite or updating feel.
  - **Utilization of the CPU**:
    - To get the number of CPU core, `sysconf(_SC_NPROCESSORS_ONLN)` function from **<unistd.h>** is required.
    - **/proc/stat** file provide the key information for total CPU usage calculate by following formula
      - **totalCPU = user + nice + system + idle + iowait + irq + softirq**
      - - - idle time: *Ii* = *idlei* where i represents a sampling time; hence the CPU utilization at time i is given by U_i = T_i - i_i
      - **CPU Usage = (U2-U1)/(totalCPU2 - totalCPU1) * 100**
  - **Utilization of Memory**
    - To get the memory_used info (physical and virtual), `struct sysinfo` defined in **<sys/sysinfo.h>** is required

    - total physical memory = total ram
    - total virtual memory = total ram + total swap space
    - used swap space = total swap space - free swap space
    - used physical memory = total ram - free ram
    - used virtual memory = used physical memory + used swap space
  - if the **--graphics** or **-g** flag is used, generate a graphical representation showing the variation of memory used
    - Graph for Memory Utilization:
      `::::::@ total relative negative change`
      `#######* total relative positive change`
      `(OPTIONAL)`
      `|o zero+`
      `|@ zero-`
    - Graph for CPU utilization:
      `|||| positive percentage increase`
- **system information**
  - To get the system information, I used `struct utsname` defined in **<sys/utsname.h>**. `uname()` function is required
    - `sysname`: the system name
    - `nodename`: the machine name
    - `version`: the version of the operation system
    - `release`: the release of the operating system
    - `machine`: the machine's architecture
    - `uptime`: time that the system has been running since its last reboot
      - **uptime** information is found in **/proc/uptime** file

Function Overview

Function	Description
`typedef struct CpuUsage`	Struct that used to store CPU Usage, contained `totalCpu` and `idle` and `char graphic[1024]` which used to store the graphic representation of CPU Utilization
`typedef struct MemoryUsage`	Struct that used to store Memory Usage, contained `phys_mem_used` and `total_ram` and `vir_mem_used` and `total_memory` and `deltaMemory` and `char output[1024]` which used to store the output of the Memory Usage

Function	Description
`MemoryUsage *newMemoryUsage()`	Allocate a space for a Memory Usage, fetch Memory Usage information at that calling time, store these information to the component in the MemoryUsage struct. Return a pointer to a MemoryUsage.
`CpuUsage *newCpuUsage()`	Allocate a space for a Cpu Usage, fetch Cpu Usage information at that calling time, store these information to the component in the CpuUsage struct. Return a pointer to a CpuUsage.
`float calculateCPUUsage(CpuUsage *cpu1, CpuUsage *cpu2)`	Calculate the CPU Usage by two `CpuUsage` struct. Return a float number that is the average of the CPU Usage by given cpu1 and cpu2
`char * displayCPUUsage(CpuUsage **cpuArray, int i, int sample, bool graphflag, bool sequentialflag, bool display, CpuUsage *cpu)`	Printout the CPU Utilization, based on the value of graphflag (`--graphics` or `-g`) or the `sequentialflag`, if graphflag holds true, `strcpy` graph to `graphic` in the CpuUsage, out put will change based on the `graphflag` and `sequentialflag`, and storing the correct output into a char array, return a char pointer.
`void MemoryUsageGraphic(MemoryUsage *memory, float last_phys_mem_used, int i)`	Update the `output` string in the given MemoryUsage *memory, `strcat` the graph to the output
`char * displayMemoryUsage(MemoryUsage **memory, int i, int sample, bool graphflag, bool sequentialflag, bool display)`	Printout the Memory Utilization, based on the value of graphflag (`--graphics` or `-g`) or the `sequentialflag`, out put will change based on the `graphflag` and `sequentialflag` and storing the correct output into a char array, return a char pointer.
`bool validInteger(char *s)`	Return True if the given string can be convert to an integer, return False if the given string can not be convert to an integer,
`int positionSampleTdelay(int argc, char **argv)`	Return an positive integer that indicate the first position of short way to showing Sample Tdelay, return 0 if no sample Tdelay short way command line argument is used.

Function	Description
`void displayRunningParameters(int samples, int tdelay, int iteration, bool sequentialflag)`	Printout the Running Parameters in a correct structure, if sequentialflag holds true, than printout the printing iteration
`char * displayUserUsage(bool display)`	Printout the UserUsage in a correct structure, and storing the correct output into a char array, return a char pointer.
`double rebootTime()`	Return the time that the system has been running since its last reboot in seconds
`void uptimeFormat(double totalSeconds)`	Printout the time that the system has been running since its last reboot in days, hours, minutes and seconds
`void displaySystemInfo()`	Print out the System Information, including the system name, machine name, OS version, OS release, machine architecture, and Printout the time that the system has been running since its last reboot in days, hours, minutes and seconds
`bool checkInput(int argc, char **argv,bool *systemflag, bool *userflag, bool *graphflag, bool *sequentialflag, bool *sampleflag, bool *tdelayflag,int *sample, int *tdelay)`	Check the command line arguments, if flag is found, make the flag be true
`int positionSampleNoTdelay(int argc, char **argv)`	Return an positive integer that indicate the first position of short way to showing Sample , return 0 if no sample short way command line argument is used.

Running the Program

1. Navigate to the directory in which `stats_function.c` and `stats_main.c` is saved on your machine (i.e. `cd`) should be in Linux OS
2. In the terminal, enter `make` to compile the program
3. To execute the program, enter `./stats` into the terminal. You may also use the following command line arguments

Flag	Description
`--system`	to print system usage report
`--user`	to print users usage report

Flag	Description
`--graphics` or `-g`	to graphically print the system stats report
`--sequential`	to sequentially print the system stats report
`--samples=N`	to indicate that the system stats will be collected and printed by `N` times, `N` is a positive number, the default value of sample is 10
`--tdelay=T`	to indicate the system stats will be collected and printed by `T` seconds, where `T` is a positive number, the default value of tdelay is 1
`x y`	where x and y are positive integer, it is a short form to write `--samples=x` and `--tdelay=y`

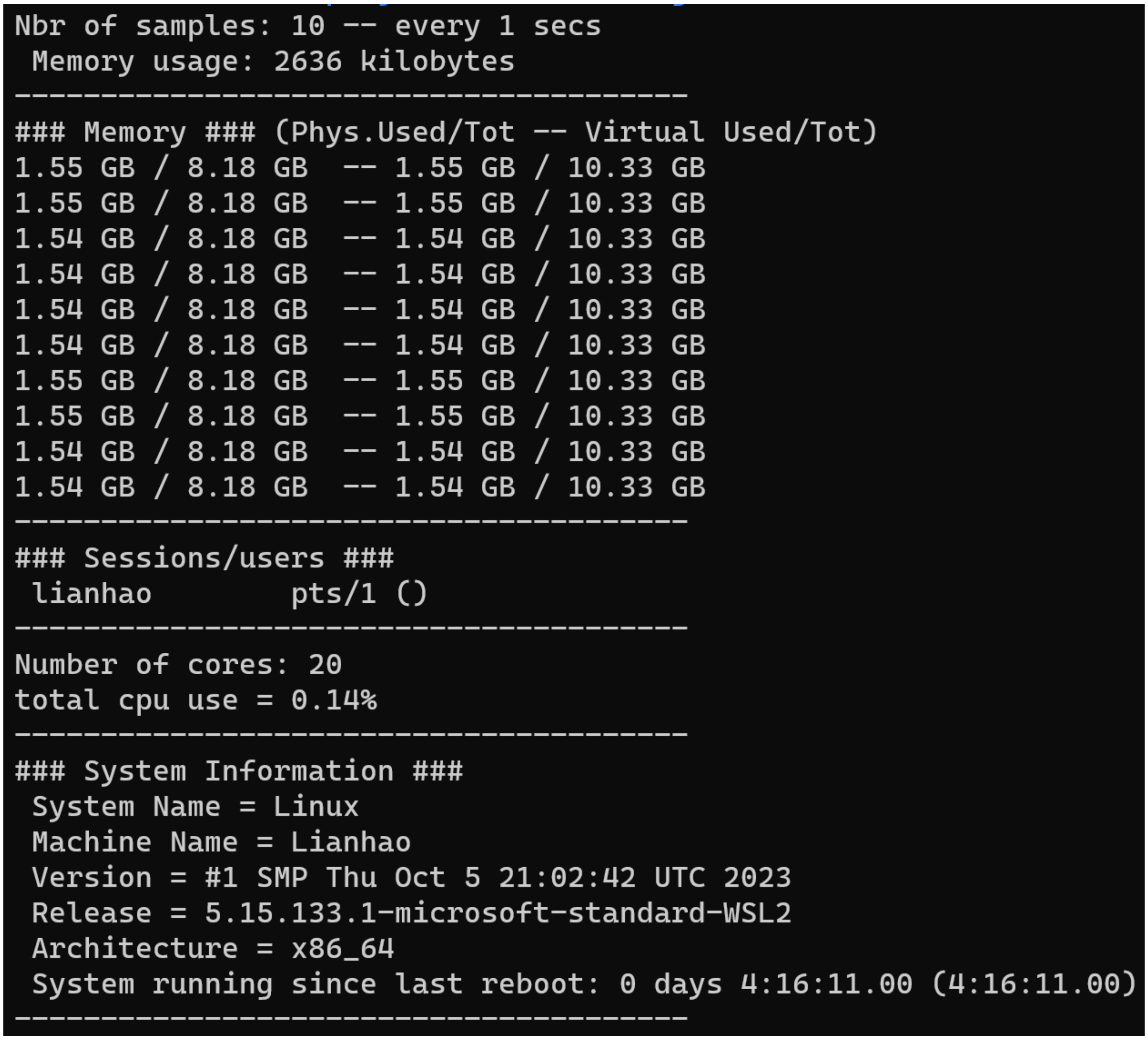
- For example
  - `./stats --system`
  - `./stats --user`
  - `./stats --graphics` or `./stats -g`
  - `./stats --sequential`
  - `./stats --samples=8`
  - `./stats --tdelay=2`
  - `./stats 8 2`

  Moreover, you can write a combination of the flag, ex) `./stats --system -g 8 2` which will graphically print out the system usage report with samples size = 8 and tdelay = 2
  - the positional arguments `x y` can be located anywhere along the input as long as the two numbers are adjacent (ex) `./stats 6 --user 3`, which is invalid, it will not update the sample value and tdelay value, they still hold the default value

  understand of graph symbol
  - Graph for Memory Utilization:
  `::::::@ total relative negative change`
  `######* total relative positive change`
  `(OPTIONAL)`
  `|o zero+`
  `|@ zero-`
  - Graph for CPU utilization:
  `|||| positive percentage increase`

4. If `Invalid Input, check your Arguments` show, means you enter invalid flag in step 3, go back to step 3 and try again with correct input command line arguments

The output of `./stats`



```
Nbr of samples: 10 -- every 1 secs
 Memory usage: 2636 kilobytes
---------------------------------------
### Memory ### (Phys.Used/Tot -- Virtual Used/Tot)
1.55 GB / 8.18 GB  -- 1.55 GB / 10.33 GB
1.55 GB / 8.18 GB  -- 1.55 GB / 10.33 GB
1.54 GB / 8.18 GB  -- 1.54 GB / 10.33 GB
1.54 GB / 8.18 GB  -- 1.54 GB / 10.33 GB
1.54 GB / 8.18 GB  -- 1.54 GB / 10.33 GB
1.54 GB / 8.18 GB  -- 1.54 GB / 10.33 GB
1.55 GB / 8.18 GB  -- 1.55 GB / 10.33 GB
1.55 GB / 8.18 GB  -- 1.55 GB / 10.33 GB
1.54 GB / 8.18 GB  -- 1.54 GB / 10.33 GB
1.54 GB / 8.18 GB  -- 1.54 GB / 10.33 GB
---------------------------------------
### Sessions/users ###
 lianhao        pts/1 ()
---------------------------------------
Number of cores: 20
total cpu use = 0.14%
---------------------------------------
### System Information ###
 System Name = Linux
 Machine Name = Lianhao
 Version = #1 SMP Thu Oct 5 21:02:42 UTC 2023
 Release = 5.15.133.1-microsoft-standard-WSL2
 Architecture = x86_64
 System running since last reboot: 0 days 4:16:11.00 (4:16:11.00)
---------------------------------------
```