How did I solve the problem?

To begin with, I had first opened each of the given auxiliary standard library to get an idea of what I was to be looking for. Taking my time with each of the libraries I found a couple things that would be very useful. This includes the RUSAGE_SELF, getrusage to find the memory utilization. The next was ut_line, ut_host, ut_user which was used to find the user and their information to present in the user section. Next was to find the system information I used the utsname struct which gave me access to the, OS name, version, machine, node_name which is basically the machine name, the version and the release. Then in the sysinfo struct it gave me a way to find the virtual memory and the physical memory. And using the struct I was able to find the used ram (physical) by subtracting the total - free and vice versa for the virtual memory. Using these I started putting together an idea about how I was going to tackle the problem, this included brainstorming ideas as to how the refreshing will work and how I could possibly print out the information in a given order. Then I started to extract the flags like sequential and such so that I could change the output based on the flags. This was done using an integer value that represented the 1 if the flag was called upon or 0 otherwise. Then since I knew that refreshing was to be done, I could simply have a for loop that runs x amount of times and refresh at the end or print out the iteration based on whether sequential or not. After the refresh was working correctly, I had to add specific if statements for each of the cases, ie. no flags, change in samples or seconds, etc. Then I had to consider forking and piping, with a little bit of research, I found out how piping works and I already had an idea as to how to use forking. So getting started, I created a nesting forking system, which would allow me to create 3 different children, which can communicate with the parent. Then I had to change the functionality of the functions to only gather and send data when in child process, and read when in parent.

function name: first_part
Parameters: - int samples: the number of samples taken - int seconds: the frequency of sampling in seconds Return value: none Summary: This function prints out the number of samples taken and the frequency of sampling in seconds. It also prints the amount of memory utilized in KB.

function name: cpuUsage
Parameters: - float* total_time: a pointer to a float variable for the total time - float* idle_time: a pointer to a float variable for the idle time
Return value: none
Summary: This function reads the /proc/stat file and retrieves the values for user, nice, system, idle, IOwait, irq, and softirq. It then calculates the total CPU time by adding these values and

sets the value of the pointer to total_time to the total CPU time and the value of the pointer to idle_time to the value of idle.

function name: core_usage
Parameters: - float old_total: the previous total CPU time - float old_idle: the previous idle CPU time - int i: the index of the current iteration - int graphics_check: a flag for checking graphics - int sequence_check: a flag for checking sequence - int pipe: a pipe for writing the results
Return value: none
Summary: This function calculates the total CPU usage by finding the CPU usage of the current iteration and the previous iteration using the cpuUsage function. It then calculates the difference between the two and calculates the overall change. The value of the CPU usage is then written to the pipe.

function name: print_core_usage
Parameters: - float (*cpu_utilization)[3]: a pointer to a 2D array for the CPU utilization - int i: the index of the current iteration - int graphics_check: a flag for checking graphics - int sequence_check: a flag for checking sequence
Return value: none
Summary: This function prints out the total CPU usage in percentage form. If graphics_check is equal to 1, it also prints out the CPU usage for each iteration using a bar graph. If sequence_check is equal to 1, it also prints out the bar graph with no additional text.

function name: system_info
Parameters: none
Return value: none
Summary: This function prints out the system information, including the system name, machine name, version, release, and architecture.

function name: session_users
Parameters: - int pipe: a pipe for writing the results
Return value: none Summary: This function prints out the sessions/users currently logged in. It reads from the /var/run/utmp file and prints out the username and terminal for each user that is currently logged in.

Function name: print_session_users
Parameters: int pipe
Return value: void

Summary: This function reads data from a given pipe and prints it to the console, until there is no more data to be read. It also adds a separator line after printing.

Function name: memory_util
Parameters: int samples, int seconds, float *phyical_memory, float *virtual_memory, int sequence_check, int graphics_check, int i, int pipe
Return value: void
Summary: This function gathers system information using the sysinfo function, calculates the physical and virtual memory usage, and writes the memory usage data to a given pipe. It also stores the memory usage data to the corresponding memory arrays.

Function name: print_memory_util
Parameters: int samples, int seconds, float *phyical_memory, float *virtual_memory, int sequence_check, int graphics_check, int i, mem memory
Return value: void
Summary: This function prints the physical and virtual memory usage data that was previously gathered and stored in the given memory arrays and memory structure. The data is printed in a formatted manner, and a separator line is added after printing. The function can be used to print a single sample or all samples depending on the sequence_check parameter. It also optionally prints a graphic representation of the virtual memory usage if the graphics_check parameter is set to 1.

function name: signal_handler
Parameters: int sig
Return value: void
Summary: A signal handler function that handles the signals SIGINT and SIGTSTP. It prompts the user to quit or resume the program depending on the user's input.

function name: handle_sigtstp
Parameters: int sig
Return value: void
Summary: A signal handler function that handles the SIGTSTP signal. It prints a message indicating that the program cannot run in the background while running interactively.

function name: main
Parameters: int argc, char *argv[]

Return value: int
Summary: The main function of the program. It reads command-line arguments, calculates the number of cores of the machine, creates three arrays for storing memory and CPU utilization values, and creates child processes for memory, user, and CPU information. It also registers signal handlers for SIGINT and SIGTSTP, and prints an error message if the number of samples or seconds is negative.

How to run the program:

In order to run the program you have to first compile the code. Then you can use any of the following (./a.out):

- without any flags to print out everything
- add numbers in the end with spaces or without spaces to only have samples size change (ie. 2 3 which is 2 samples every 3 seconds or just 3 to have 3 samples for default seconds value)
- Use —samples= or —tdelay= to use different amount of samples or different times between each sample
- use —graphics to show graphics for the memory utilization and cpu usage
- use —sequential to show the output in a chronological formate where each iteration is a new output
- use —user to print out the users that are currently running on the system(BV machine)
- use —system to print out the memory utilization and the cpu utilization
- _____
- Also use Control C to stop the code at any time
- Use Control Z to do nothing