COS 333 – Advanced Programming Techniques

Programming Assignment III

Assigned: Wednesday, March 13, 1996

Due: 11:59pm, Monday, April 1, 1996

Program: Many programs suffer from *space leaks*—memory storage that becomes inaccessible because it is not `free`d when no longer in use. At best, space leaks degrade performance; at worst, space leaks cause serious crashes.

You are to design a graphical *memory gauge* that dynamically plots over time the amount of memory in use by a long-running program. To do this, you will use wrapper functions for `malloc`[1] and `free` that keep the size of an allocated block with the block itself. A timer will interrupt the program being monitored at one second intervals and send the number of bytes `malloc`'d and `free`d during the previous second to a graphical shell script. This script plots the amount of memory currently in use.

We supply the following in `~cs333/src/memory-gauge/`:

`my-mem.h` – A C header file defining the interface to the wrapper functions for `malloc` and `free`.

`my-mem.c` – Skeletons for the wrapper C functions `my_malloc` and `my_free` that "replace" `malloc` and `free`. The wrapper functions keep the size of an allocated object with the object so that it is possible to know how many bytes of storage a call to `free` reclaims. The additional function `my_mem_init` is called immediately on program start; you may use it to initialize things such as timers.

`driver.c` – A sample program to test your memory gauge.[2] It performs various amounts of somewhat random storage allocation and deallocation. This program makes calls to `my_malloc`, `my_free`, and `my_mem_init`; it must therefore be linked with `my-mem.o`.

`simple-timer.c` – A stand-alone C program that illustrates the use of timers to periodically interrupt regular execution.

`makefile` – A `makefile` to build the above object files and programs.

For this assignment, you must:

(i) combine `my-mem.c` and `simple-timer.c` so that the amount of allocation and deallocation (in bytes) that occurred in the last second of execution can be computed.

(ii) design a graphical gauge (using, *e.g.*, `Tcl/Tk` or `Tksh`) that plots the amount of `malloc`'d memory in use over time.

(iii) devise a mechanism for communicating the amount of allocation/deallocation to the gauge every second. Some possibilities are via a fifo (see `man mkfifo`), via `stderr`, or by forking a process for the gauge from `my-mem.c` and communicating with it via a pipe. Document the advantages and disadvantages of your selected approach.

(iv) incorporate some mechanism to rescale your gauge to accommodate significant changes in memory consumption.[3]

Hint: Build and test (i) and (ii) separately before hooking them up (iii).

Submission Procedure: Along with your program(s), submit a `makefile` and a `README` file; see the course's web page for details.

[1] You need not handle the other memory library functions, although doing so would make your gauge more precise and hence more reliable.

[2] You may want to design additional test drivers or link your memory gauge with real C programs.

[3] The `xload` program uses an adequate rescaling mechanism.