Exercise 16.2: Invoking the OOM Killer

- When the **Linux** kernel gets under extreme memory pressure it invokes the dreaded **OOM** (**O**ut **O**f **M**emory) **Killer**. This tries to select the “best” process to kill to help the system recover gracefully.
- We are going to force the system to run short on memory and watch what happens. The first thing to do is to open up a terminal window, and in it type:

  ```
  $ sudo tail -f /var/log/messages
  ```

  in order to watch kernel messages as they appear.

- An even better way to look is furnished by:

  ```
  $ dmesg -w
  ```

  as it does not show non-kernel messages.

- This exercise will be easier to perform if we turn off all swap first with the command:

  ```
  $ sudo /sbin/swapoff -a
  ```

  Make sure you turn it back on later with

  ```
  $ sudo /sbin/swapon -a
  ```

- Now we are going to put the system under increasing memory pressure. You are welcome to find your own way of doing it but we also supply a program for consuming the memory:

lab_wastemem.c

```c
/* simple program to defragment memory, J. Cooperstein 2/04
 */

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/sysinfo.h>
#include <signal.h>

#define MB (1024*1024)
#define BS 16                           /* will allocate BS*MB at each step */
#define CHUNK (MB*BS)
#define QUIT_TIME 20
void quit_on_timeout(int sig)
{
        printf("\n\nTime expired, quitting\n");
        exit(EXIT_SUCCESS);
}

int main(int argc, char **argv)
{
        struct sysinfo si;
        int j, m;
```

```c
        char *c;

        /* get total memory on the system */
        sysinfo(&si);
        m = si.totalram / MB;
        printf("Total System Memory in MB = %d MB\n", m);
        m = (9 * m) / 10;         /* drop 10 percent */
        printf("Using somewhat less: %d MB\n", m);

        if (argc == 2) {
                m = atoi(argv[1]);
                printf("Choosing instead mem = %d MB\n", m);
        }

        signal(SIGALRM, quit_on_timeout);
        printf("Will quite in QUIT_TIME seconds if no normal termination\n");
        alarm(QUIT_TIME);

        for (j = 0; j <= m; j += BS) {
                /* yes we know this is a memory leak, no free,
                 * that's the idea!
                 */
                c = malloc(CHUNK);
                /* just fill the block with j over and over */
                memset(c, j, CHUNK);
                printf("%8d", j);
                fflush(stdout);
        }
        printf("\n\n    Sleeping for 5 seconds\n");
        sleep(5);
        printf("\n\n    Quitting and releasing memory\n");
        exit(EXIT_SUCCESS);
}
```

It takes as an argument how many MB to consume. Keep running it, gradually increasing the amount of memory requested until your system runs out of memory.

Please Note

You should be able to compile the program and run it by just doing:

```
$ gcc -o lab_wastemem lab_wastemem.c
$ ./lab_wastemem 4096
```

which would waste 4 GB. It would be a good idea to run **gnome-system-monitor** or another memory monitoring program while it is running (although the display may freeze for a while!)

- You should see the **OOM** (Out of Memory) killer swoop in and try to kill processes in a struggle to stay alive. Who gets clobbered first?

✓ Solution 16.2

```
Please see SOLUTIONS/s_16/lab_wastemem.c
Please see SOLUTIONS/s_16/lab_waste.sh
```