Chapter 13

Question 1:

```
FREE(1)                    User Commands                   FREE(1)
NAME
     free - Display amount of free and used memory in the system
SYNOPSIS
     free [options]
DESCRIPTION
     free  displays the total amount of free and used physical and swap mem-
     ory in the system, as well as the buffers and caches used by  the  ker-
     nel.  The  information  is  gathered by parsing /proc/meminfo. The dis-
     played columns are:
     total  Total installed memory (MemTotal and SwapTotal in /proc/meminfo)
     used   Used memory (calculated as total - free - buffers - cache)
     free   Unused memory (MemFree and SwapFree in /proc/meminfo)
     shared Memory used (mostly) by tmpfs (Shmem in /proc/meminfo)
     buffers
          Memory used by kernel buffers (Buffers in /proc/meminfo)
     cache  Memory used by the page cache and slabs (Cached and SReclaimable
          in /proc/meminfo)
     buff/cache
          Sum of buffers and cache
     available
          Estimation  of  how  much  memory  is available for starting new
          applications, without swapping. Unlike the data provided by  the
          cache  or  free fields, this field takes into account page cache
          and also that not all reclaimable memory slabs will be reclaimed
          due to items being in use (MemAvailable in /proc/meminfo, avail-
          able on kernels 3.14, emulated on kernels 2.6.27+, otherwise the
          same as free)
```

Question 2:

```
[zw335812@login-students ~]$ free -m
          total       used       free     shared  buff/cache   available
Mem:       160644       4417      91161       1066      65064      153708
Swap:        2047          0       2047
```

Question 3:

memory-user.c file is in github

Question 4:

When the memory-user program runs, the output of the free command shows a clear increase in the “used” memory and a corresponding decrease in the “free” memory, roughly matching the amount of memory allocated by the program. This happens because the program actively touches each page of its allocated memory, forcing the operating system to commit physical RAM to the process. The “available” memory value drops as well, showing that less RAM is left for other applications while the program runs.

After the program is killed, the memory usage immediately returns to its original levels and used memory decreases and free memory increases—since the operating system reclaims the memory once the process terminates. When testing larger allocations, the system may slow down or begin using swap space, and extremely large requests can cause allocation failures if the requested memory exceeds physical RAM. These results align with expectations and demonstrate how Linux dynamically manages memory allocation and reclamation for active processes.

Question 5:

```
[zw335812@login-students homework4]$ man pmap
PMAP(1)                    User Commands                    PMAP(1)
NAME
       pmap - report memory map of a process
SYNOPSIS
       pmap [options] pid [...]
DESCRIPTION
       The pmap command reports the memory map of a process or processes.
OPTIONS
       -x, --extended
              Show the extended format.
       -d, --device
              Show the device format.
       -q, --quiet
              Do not display some header or footer lines.
       -A, --range low,high
              Limit  results to the given range to low and high address range.
              Notice that the low and high arguments are single  string  sepa-
              rated with comma.
       -X     Show  even  more  details  than  the  -x option. WARNING: format
              changes according to /proc/PID/smaps
       -XX    Show everything the kernel provides
       -p, --show-path
              Show full path to files in the mapping column
       -c, --read-rc
              Read the default configuration
       -C, --read-rc-from file
              Read the configuration from file
       -n, --create-rc
              Create new default configuration
       -N, --create-rc-to file
              Create new configuration to file
       -h, --help
              Display help text and exit.
       -V, --version
              Display version information and exit.
EXIT STATUS
              0      Success.
              1      Failure.
              42     Did not find all processes asked for.
```

```
SER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
zw335812 3173922  0.0  0.0  90268 10224 ?        Ss   17:28   0:00 /usr/lib/systemd/systemd --user
zw335812 3173926  0.0  0.0 335152  8996 ?        S    17:28   0:00 (sd-pam)
zw335812 3173941  0.0  0.0 165280  7024 ?        S    17:28   0:00 sshd: zw335812@pts/62
zw335812 3173942  0.0  0.0  27636  5508 pts/62   Ss   17:28   0:00 -bash
zw335812 3176148 15.5  0.0 106848 103624 pts/62  S    17:37   0:01 ./memory-user 100
zw335812 3176154  0.0  0.0  58848  3860 pts/62   R+   17:37   0:00 ps ux
```

Question 6

```
[zw335812@login-students homework4]$ pmap 3176325
3176325:   ./memory-user 10
0000000000400000      4K r-x-- memory-user
0000000000600000      4K r---- memory-user
0000000000601000      4K rw--- memory-user
0000000001673000    132K rw---   [ anon ]
00007f571c77d000  10244K rw---   [ anon ]
00007f571d17e000   1844K r-x-- libc-2.28.so
00007f571d34b000   2048K ----- libc-2.28.so
00007f571d54b000     16K r---- libc-2.28.so
00007f571d54f000      8K rw--- libc-2.28.so
00007f571d551000     16K rw---   [ anon ]
00007f571d555000    188K r-x-- ld-2.28.so
00007f571d772000      8K rw---   [ anon ]
00007f571d784000      4K r---- ld-2.28.so
00007f571d785000      8K rw--- ld-2.28.so
00007ffd9d751000    136K rw---   [ stack ]
00007ffd9d7a9000     16K r----   [ anon ]
00007ffd9d7ad000      8K r-x--   [ anon ]
ffffffffff600000      4K r-x--   [ anon ]
 total            14692K
```

Address Range	Size	Meaning
0000000000400000 → 0000000000601000	~12 KB	The program's code and data segments — where your compiled binary lives.
0000000001673000 and 00007f571c77d000	~132 KB and 10,244 KB	Anonymous ([anon]) memory, which comes from malloc() — this is your program's allocated heap! This matches your 10 MB request.
libc-2.28.so, ld-2.28.so	A few MB	Shared libraries loaded from disk — standard C library and dynamic loader.
[stack]	~136 KB	The stack used for local variables and function calls.
[anon] (small segments)	A few KB	Internal OS bookkeeping or thread-local data.

Question 7

```
[zw335812@login-students homework4]$ pmap 3176325 -x
3176325:   ./memory-user 10
Address           Kbytes     RSS   Dirty Mode  Mapping
0000000000400000       4       4       0 r-x-- memory-user
0000000000600000       4       4       4 r---- memory-user
0000000000601000       4       4       4 rw--- memory-user
0000000001673000     132       4       4 rw---   [ anon ]
00007f571c77d000   10244   10240   10240 rw---   [ anon ]
00007f571d17e000    1844    1160       0 r-x-- libc-2.28.so
00007f571d34b000    2048       0       0 ----- libc-2.28.so
00007f571d54b000      16      16      16 r---- libc-2.28.so
00007f571d54f000       8       8       8 rw--- libc-2.28.so
00007f571d551000      16      12      12 rw---   [ anon ]
00007f571d555000     188     172       0 r-x-- ld-2.28.so
00007f571d772000       8       8       8 rw---   [ anon ]
00007f571d784000       4       4       4 r---- ld-2.28.so
00007f571d785000       8       8       8 rw--- ld-2.28.so
00007ffd9d751000     136      16      16 rw---   [ stack ]
00007ffd9d7a9000      16       0       0 r----   [ anon ]
00007ffd9d7ad000       8       4       0 r-x--   [ anon ]
ffffffffff600000       4       0       0 r-x--   [ anon ]
---------------- ------- ------- -------
total kB           14692   11664   10324
```

RSS *Resident Set Size*: how much of that region is actually in physical RAM.

Dirty How much memory has been modified (and not shared or file-backed).

Mode Read/Write/Execute permissions (r-x--, rw---, etc.).

Question 8

When running pmap on the memory-user program with different memory sizes, the output shows the memory map of the process, so all the regions of memory it's using code/stack/heap. The most noticeable part is the heap segment, which grows larger as you increase the number of megabytes passed to the program. For example, when running ./memory-user 100, the heap show around 100 MB of anonymous memory, while ./memory-user 500 shows roughly 500 MB.

This matches expectations because our program allocates a large contiguous block of memory using malloc(), which resides in the heap. The rest of the mappings (like stack and shared libraries) remain nearly the same regardless of how much memory we allocate. Thus, pmap confirms that the allocated memory is indeed reserved by the process and that the heap size scales consistently with the requested amount.

Chapter 14

Question 1:
null.c in github

Question 2:

```
(gdb) run
Starting program: /home/zw335812/cs5600/homework4/null
About to dereference a NULL pointer...
Program received signal SIGSEGV, Segmentation fault.
0x00000000004005f4 in main () at null.c:9
9	    printf("Value: %d\n", *ptr);
Missing separate debuginfos, use: yum debuginfo-install glibc-2.28-251.el8_10.22.x86_64
```

Question 3:

```
[zw335812@login-students homework4]$ valgrind --leak-check=yes ./null
==3185714== Memcheck, a memory error detector
==3185714== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==3185714== Using Valgrind-3.22.0 and LibVEX; rerun with -h for copyright info
==3185714== Command: ./null
==3185714==
About to dereference a NULL pointer...
==3185714== Invalid read of size 4
==3185714==    at 0x4005F4: main (null.c:9)
==3185714==  Address 0x0 is not stack'd, malloc'd or (recently) free'd
==3185714==
==3185714==
==3185714== Process terminating with default action of signal 11 (SIGSEGV)
==3185714==  Access not within mapped region at address 0x0
==3185714==    at 0x4005F4: main (null.c:9)
==3185714==  If you believe this happened as a result of a stack
==3185714==  overflow in your program's main thread (unlikely but
==3185714==  possible), you can try to increase the size of the
==3185714==  main thread stack using the --main-stacksize= flag.
==3185714==  The main thread stack size used in this run was 8388608.
==3185714==
==3185714== HEAP SUMMARY:
==3185714==     in use at exit: 1,024 bytes in 1 blocks
==3185714==   total heap usage: 1 allocs, 0 frees, 1,024 bytes allocated
==3185714==
==3185714== LEAK SUMMARY:
==3185714==    definitely lost: 0 bytes in 0 blocks
==3185714==    indirectly lost: 0 bytes in 0 blocks
==3185714==      possibly lost: 0 bytes in 0 blocks
==3185714==    still reachable: 1,024 bytes in 1 blocks
==3185714==         suppressed: 0 bytes in 0 blocks
==3185714== Reachable blocks (those to which a pointer was found) are not shown.
==3185714== To see them, rerun with: --leak-check=full --show-leak-kinds=all
==3185714==
==3185714== For lists of detected and suppressed errors, rerun with: -s
==3185714== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
Segmentation fault (core dumped)
```

Question 4

```
(gdb) run
Starting program: /home/zw335812/cs5600/homework4/memory_leak
Allocating memory for 100 integers...
First element: 0
Last element: 198
Exiting program without freeing memory...
[Inferior 1 (process 3186727) exited normally]
Missing separate debuginfos, use: yum debuginfo-install glibc-2.28-251.el8_10.22.x86_64
```

```
[zw335812@login-students homework4]$ valgrind --leak-check=yes ./memory_leak
==3186503== Memcheck, a memory error detector
==3186503== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==3186503== Using Valgrind-3.22.0 and LibVEX; rerun with -h for copyright info
==3186503== Command: ./memory_leak
==3186503==
Allocating memory for 100 integers...
First element: 0
Last element: 198
Exiting program without freeing memory...
==3186503==
==3186503== HEAP SUMMARY:
==3186503==     in use at exit: 400 bytes in 1 blocks
==3186503==   total heap usage: 2 allocs, 1 frees, 1,424 bytes allocated
==3186503==
==3186503== 400 bytes in 1 blocks are definitely lost in loss record 1 of 1
==3186503==    at 0x4C39185: malloc (vg_replace_malloc.c:442)
==3186503==    by 0x400659: main (memory_leak.c:11)
==3186503==
==3186503== LEAK SUMMARY:
==3186503==    definitely lost: 400 bytes in 1 blocks
==3186503==    indirectly lost: 0 bytes in 0 blocks
==3186503==      possibly lost: 0 bytes in 0 blocks
==3186503==    still reachable: 0 bytes in 0 blocks
==3186503==         suppressed: 0 bytes in 0 blocks
==3186503==
==3186503== For lists of detected and suppressed errors, rerun with: -s
==3186503== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

Question 5

end_zero.c is in github

```
[zw335812@login-students homework4]$ ./end_zero
Program completed successfully
```

```
[zw335812@login-students homework4]$ valgrind --leak-check=yes ./end_zero
==3187274== Memcheck, a memory error detector
==3187274== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==3187274== Using Valgrind-3.22.0 and LibVEX; rerun with -h for copyright info
==3187274== Command: ./end_zero
==3187274==
==3187274== Invalid write of size 4
==3187274==    at 0x4006F2: main (end_zero.c:14)
==3187274==  Address 0x52231d0 is 0 bytes after a block of size 400 alloc'd
==3187274==    at 0x4C39185: malloc (vg_replace_malloc.c:442)
==3187274==    by 0x4006B7: main (end_zero.c:6)
==3187274==
Program completed successfully
==3187274==
==3187274== HEAP SUMMARY:
==3187274==     in use at exit: 0 bytes in 0 blocks
==3187274==   total heap usage: 2 allocs, 2 frees, 1,424 bytes allocated
==3187274==
==3187274== All heap blocks were freed -- no leaks are possible
==3187274==
==3187274== For lists of detected and suppressed errors, rerun with: -s
==3187274== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

Question 6

print_after_free.c is in github

it does run without any issue

```
[zw335812@login-students homework4]$ ./print_after_free
Value at data[50]: 100
```

Valgrind can detect the problem

```
[zw335812@login-students homework4]$ valgrind --leak-check=yes ./print_after_free
==3187642== Memcheck, a memory error detector
==3187642== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==3187642== Using Valgrind-3.22.0 and LibVEX; rerun with -h for copyright info
==3187642== Command: ./print_after_free
==3187642==
==3187642== Invalid read of size 4
==3187642==    at 0x400718: main (print_after_free.c:21)
==3187642==  Address 0x5223108 is 200 bytes inside a block of size 400 free'd
==3187642==    at 0x4C3C4CB: free (vg_replace_malloc.c:985)
==3187642==    by 0x40070D: main (print_after_free.c:18)
==3187642==  Block was alloc'd at
==3187642==    at 0x4C39185: malloc (vg_replace_malloc.c:442)
==3187642==    by 0x4006B7: main (print_after_free.c:6)
==3187642==
Value at data[50]: 100
==3187642==
==3187642== HEAP SUMMARY:
==3187642==     in use at exit: 0 bytes in 0 blocks
==3187642==   total heap usage: 2 allocs, 2 frees, 1,424 bytes allocated
==3187642==
==3187642== All heap blocks were freed -- no leaks are possible
==3187642==
==3187642== For lists of detected and suppressed errors, rerun with: -s
==3187642== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

Question 7

free_middle.c in the github

```
[zw335812@login-students homework4]$ ./free_middle
Allocated memory at: 0x7f32a0
Middle of array at: 0x7f3368
free(): invalid pointer
Aborted (core dumped)
```

```
[zw335812@login-students homework4]$ valgrind --leak-check=yes ./free_middle
```

```
==3188324== Memcheck, a memory error detector
==3188324== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==3188324== Using Valgrind-3.22.0 and LibVEX; rerun with -h for copyright info
==3188324== Command: ./free_middle
==3188324==
Allocated memory at: 0x5223040
Middle of array at: 0x5223108
==3188324== Invalid free() / delete / delete[] / realloc()
==3188324==    at 0x4C3C4CB: free (vg_replace_malloc.c:985)
==3188324==    by 0x400795: main (free_middle.c:21)
==3188324==  Address 0x5223108 is 200 bytes inside a block of size 400 alloc'd
==3188324==    at 0x4C39185: malloc (vg_replace_malloc.c:442)
==3188324==    by 0x400707: main (free_middle.c:6)
==3188324==
Program completed
==3188324==
==3188324== HEAP SUMMARY:
==3188324==     in use at exit: 400 bytes in 1 blocks
==3188324==   total heap usage: 2 allocs, 2 frees, 1,424 bytes allocated
==3188324==
==3188324== 400 bytes in 1 blocks are definitely lost in loss record 1 of 1
==3188324==    at 0x4C39185: malloc (vg_replace_malloc.c:442)
==3188324==    by 0x400707: main (free_middle.c:6)
==3188324==
==3188324== LEAK SUMMARY:
==3188324==    definitely lost: 400 bytes in 1 blocks
==3188324==    indirectly lost: 0 bytes in 0 blocks
==3188324==      possibly lost: 0 bytes in 0 blocks
==3188324==    still reachable: 0 bytes in 0 blocks
==3188324==         suppressed: 0 bytes in 0 blocks
==3188324==
==3188324== For lists of detected and suppressed errors, rerun with: -s
==3188324== ERROR SUMMARY: 2 errors from 2 contexts (suppressed: 0 from 0)
```

Question 8

vector.c in github

```
[zw335812@login-students homework4]$ ./vector
Added 0 (size=1, capacity=2)
Added 10 (size=2, capacity=2)
Added 20 (size=3, capacity=4)
Added 30 (size=4, capacity=4)
Added 40 (size=5, capacity=8)
Added 50 (size=6, capacity=8)
Added 60 (size=7, capacity=8)
Added 70 (size=8, capacity=8)
Added 80 (size=9, capacity=16)
Added 90 (size=10, capacity=16)
Vector contents:
0 10 20 30 40 50 60 70 80 90
[zw335812@login-students homework4]$ valgrind --leak-check=full ./vector
==3264680== Memcheck, a memory error detector
==3264680== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==3264680== Using Valgrind-3.22.0 and LibVEX; rerun with -h for copyright info
==3264680== Command: ./vector
==3264680==
Added 0 (size=1, capacity=2)
Added 10 (size=2, capacity=2)
Added 20 (size=3, capacity=4)
Added 30 (size=4, capacity=4)
Added 40 (size=5, capacity=8)
Added 50 (size=6, capacity=8)
Added 60 (size=7, capacity=8)
Added 70 (size=8, capacity=8)
Added 80 (size=9, capacity=16)
Added 90 (size=10, capacity=16)
Vector contents:
0 10 20 30 40 50 60 70 80 90
==3264680==
==3264680== HEAP SUMMARY:
==3264680==     in use at exit: 0 bytes in 0 blocks
==3264680==   total heap usage: 5 allocs, 5 frees, 1,144 bytes allocated
==3264680==
==3264680== All heap blocks were freed -- no leaks are possible
==3264680==
==3264680== For lists of detected and suppressed errors, rerun with: -s
==3264680== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

There is no memory leak as I have done proper resizing when capacity doubles.