SeifEldin Abdelhamid

Professor Ying Li

CS333

February 18, 2025

Project01: Learning C Programming

Google site: https://sites.google.com/colby.edu/seifsproject1/home

Task 1:

```
char:
0: 53

short:
0: 02
1: 00

int:
0: 02
1: 00
2: 00
3: 00

long:
0: 4C
1: 00
2: 00
3: 00
4: 00
5: 00
6: 00
7: 00

float:
0: C3
1: F5
2: 48
3: 40

double:
0: 93
1: 18
2: 04
3: 56
4: 0E
5: 2D
6: 09
7: 40
```

2-

Is the machine you are using big-endian or little-endian?

The machine used is little-endian, as seen in how multi-byte data types are stored. In little-endian systems, the least significant byte (LSB) is at the smallest address, and the most significant byte (MSB) is at the largest. This is clear from the memory layout of short, int, long, float, and double in the output.

3- How does the problem output tell you?

As we know the least significant bytes are stored at the beginning of the memory. This matches the definition of little endian, where the least significant bytes come first.

Task 2:

```
0: 18
1: 94
2: 64
3: 08
4: 03
5: 00
6: 00
7: 00
8: 4E
9: 00
10: 21
11: D5
12: C3
13: F5
14: 48
15: 40
16: EF
17: CD
18: AB
19: 90
20: 78
21: 56
22: 34
23: 12
24: 50
25: 94
26: 64
27: 08
28: 03
29: 00
30: 34
31: 12
32: 37
33: 72
34: 38
35: 4E
36: 65
37: 2C
38: 09
39: 40
40: 00
41: 20
42: 36
43: 02
44: 02
45: 00
46: 00
47: 53
48: 30
49: 00
50: 00
51: 00
52: 00
53: 00
54: 00
55: 00
56: 80
57: 96
58: 64
59: 08
60: 03
61: 00
62: 00
63: 00
64: CD
65: 02
66: 2D
67: 02
68: 02
69: 00
70: 00
71: 00
72: 00
73: 00
74: 00
75: 00
76: 00
77: 00
78: 00
79: 00
80: 00
81: 00
82: 00
83: 00
84: 00
85: 00
86: 00
87: 00
88: 00
89: 00
90: 00
91: 00
92: 00
93: 00
94: 00
95: 00
96: 00
97: 00
98: 00
99: 00
100: 00
```

What seems to be the overall layout of the stack?

2-

The stack grows downwards (from high memory addresses to low memory addresses).

3- Are there any non-zero values you can't immediately make sense of?

Some non-zero values may not be immediately clear, especially in floating-point numbers and uninitialized memory regions. Floating-point values use IEEE 754 format, which can look unfamiliar in hexadecimal. Also, padding bytes in structs may contain leftover data, making some values seem random.

4- Can you find the variables defined in your C program? Highlight the ones you find and explain how you know you have found them.

In the stack memory output, we can confidently identify the variables defined in the C program based on their initialized values and memory representation, considering the machine's little-endian architecture. Here are the ones I found, but I am not really sure if they are true.

- `char c = 'S'`
- `short s = 2`
- `int i = 2`
- `long l = 76`
- `float f = 3.14`
- `double d = 3.14159`

Task 3:

```
Processes: 477 total, 3 running, 474 sleeping, 2765 threads                                                              11:34:41
Load Avg: 2.44, 2.26, 2.43  CPU usage: 16.4% user, 4.19% sys, 79.76% idle  SharedLibs: 293M resident, 48M data, 250M linkedit.
MemRegions: 73 total, 2192K resident, 0B private, 558M shared. PhysMem: 7538M used (1456M wired, 3343M compressor), 84M unused.
VM: 193T vsize, 8998M framework vsize, 2556105(0) swapins, 3473411(0) swapouts. Networks: packets: 47090506/57G in, 12316511/2064M out.
Disks: 63072470/1485G read, 16009108/314G written.

PID    COMMAND      %CPU  TIME     #TH   #WQ  #PORT MEM    PURG   CMPRS  PGRP  PPID  STATE    BOOSTS            %CPU_ME %CPU_OTHRS UID  FAULTS    COW
47095* Project_samo 100.4 02:15.92 2/1   0    18    2117K  0B     1572K  47095 46545 running  *0[1]             0.00000 0.00000    501  1118      90
3137*  WindowServer 12.2  04:44:14 19    5    4978+ 1201M+ 1600K+ 261M-  3137  1     sleeping *0[1]             0.00102 0.05806    88   28271310+ 151893
0*     kernel_task  10.8  04:38:09 568/8 0    0     513M+  0B     0B     0     0     running   0[0]             0.00000 0.00000    0    266450    0
3326*  Messages     3.8   92:40.44 11    2    1971  408M   0B     384M   3326  1     sleeping *12557[5]         0.04529 0.00000    501  10239899  1436
47132* top          3.7   00:05.76 2/1   0    38+   9897K+ 0B     3300K  47132 47101 running  *0[1]             0.00000 0.00000    0    61889+    131
1*     launchd      1.1   20:15.87 4     3    3950+ 20M+   0B     8448K- 1     0     sleeping  0[0]             0.00000 0.01099    0    2019775+  18124
357*   locationd    1.0   35:41.16 7     4    311   12M    0B     6528K  357   1     sleeping *0[529142+]       0.00006 0.01697    205  1536397   257
29794* bluetoothd   1.0   12:29.89 12    6    435   25M    0B-    18M    29794 1     sleeping *0[1]             0.04944 0.00000    0    711809    234
512*   searchpartyd 0.8   19:32.71 9     7    128   10M    0B     5088K  512   1     sleeping *0[564211+]       0.00000 0.01367    0    1251584   272
3323*  WhatsApp     0.8   44:15.14 18    3    1121  294M   0B     265M   3323  1     sleeping *860[3]           0.00017 0.00000    501  17078262  1852
3657*  Code Helper  0.7   13:23.78 21    1    253   630M+  0B     385M-  3319  3319  sleeping *0[5]             0.00000 0.00000    501  33948496+ 3049
3623*  EpicGamesLau 0.6   28:39.81 42    5    394   111M   0B     100M   3623  1     sleeping *1[2]             0.00000 0.00000    501  4464659   1562
3371*  BiomeAgent   0.6   04:14.50 3     2    216+  8401K+ 256K+  3472K- 3371  1     sleeping *0[27289+]        0.00000 0.01443    501  1693880+  266
47194* screencaptur 0.5   00:00.56 2     1    67    8770K+ 0B-    0B     3349  3349  sleeping *0[658+]          0.00045 0.00000    501  5496+     137
349*   opendirector 0.5   11:20.57 7     6    1475  11M    0B-    6512K- 349   1     sleeping *0[1]             0.00000 0.00647    0    1591343+  174
3324*  Google Chrom 0.4   05:40.30 38    2    619   104M   0B     87M-   3324  1     sleeping *0[18416+]        0.00000 0.00000    501  5511599+  2178
3395*  com.apple.We 0.4   29:35.25 11    5    164   104M-  0B     74M-   3395  1     sleeping *0[44858]         0.00000 0.00000    501  6690335+  264
3277*  Safari       0.4   47:03.60 10    3    2785+ 665M   0B     625M-  3277  1     sleeping *0[67161]         0.01921 0.00000    501  17850646+ 30627
755*   nearbyd      0.3   09:51.78 7     5    107   5953K  0B     2608K  755   1     sleeping *3[10565]         0.00000 0.00761    268  417899    176
3413*  sharingd     0.3   15:42.95 5     2    524   24M    0B     18M    3413  1     sleeping *0[1]             0.00000 0.00777    501  1025144   420
3319*  Electron     0.3   08:05.23 37    1    582   306M   0B     252M   3319  1     sleeping *1[4784]          0.00000 0.00000    501  12598706+ 1904
46753* CommCenter   0.2   00:00.36 8     4    159+  6849K+ 0B     4720K- 46753 1     sleeping *0[1]             0.00374 0.00000    501  8435+     212
8578*  ContextStore 0.2   01:19.07 3     2    103+  8081K+ 0B     1760K- 8578  1     sleeping *0[6604+]         0.00000 0.00621    501  1051188+  216
3595*  Code Helper  0.2   02:58.95 14    3    175+  184M+  0B     51M    3319  3319  sleeping *1[2]             0.00050 0.00000    501  3123176+  624
3275*  rapportd     0.2   07:28.89 2     1    261   8305K  0B     4944K- 3275  1     sleeping *0[1]             0.00000 0.00556    501  680068+   258
311*   logd         0.2   16:51.55 4     3    1574  24M    0B     29M-   311   1     sleeping *0[1]             0.00000 0.00000    0    2303125+  128
3717*  Code Helper  0.2   03:20.49 21    1    102   101M   0B     65M    3319  3319  sleeping *0[1]             0.00000 0.00000    501  1256030   519
373*   notifyd      0.1   02:26.69 3     2    833+  3953K+ 0B     880K-  373   1     sleeping *0[1]             0.00000 0.00184    0    277729+   86
42427* proactived   0.1   00:02.34 3     2    82+   5217K- 0B     2272K- 42427 1     sleeping  0[13]            0.00000 0.00429    501  57070+    192
12922* milod        0.1   04:59.11 3     2    135   9105K  0B     6640K  12922 1     sleeping *1[3]             0.00000 0.00377    501  264109    239
37427* com.apple.We 0.1   14:16.61 10    3    82    781M   0B     614M   37427 1     sleeping  0[72338+]        0.00000 0.00000    501  34210893  11682
1040*  trustd       0.1   04:42.70 2     1    119   8929K  0B-    5056K  1040  1     sleeping *0[26314+]        0.00000 0.00374    282  1249488+  174
351*   apsd         0.1   05:04.07 3     2    385+  10M+   0B     4864K- 351   1     sleeping *0[1]             0.00000 0.00290    0    1394972+  215
```

Briefly describe the memory requirements when using and not using the free statement

2-

If you don't free allocated memory, it keeps using more and more space, causing memory leaks. Using the free function helps return the memory when it's no longer needed, making sure it can be used again and keeping memory usage low.

Task 4:

```
Memory layout of struct NewStruct:
0: 53
1: 00
2: E9
3: 07
4: 40
5: 20
6: 00
7: 00
```

A- Does the size of the result match your expectation?

Yes, I believe so.

B- Are there any gaps in the way the fields of the structure are laid out?

Yes, there is a gap in the structure's layout between the char and short fields to align the short on a 2-byte boundary, confirming efficient memory access according to the architecture's alignment requirements.

Task 5:

```
Please input your name for a new bank account: Seif Abdelhamid
Thank you Seif Abdelhamid, your new account has been initialized with balance 0.
```

First, find a string that doesn't work, the string that doesn't work is gets, perhaps, I switched it to fgets and it worked fine.

Here is the memory content on my bad string:

```
Please input your name for a new bank account: Seif Abdelhamid
Thank you Seif Abdelhamid , your new account has been initialized with balance 0.

Memory layout of struct BankAccount:
53 65 69 66 20 41 62 64
65 6C 68 61 6D 69 64 20
00 00 27 02 00 00 00 00
```

What has gone wrong was, the string overflows the name array, overwriting nearby memory. Padding bytes between name and balance may contain leftover values. fgets() prevents some issues, but memory is not always cleared.

Extension 1:

I wrote a program to create a bus error by misaligning a pointer and trying to use it. When I ran it nothing happened, which was expected. This likely means my system allows misaligned memory access without causing an error. On some other systems, the program might crash instead.