OSTEP Chapter 14

ECE 3600, Fall 2022

Table of Contents

1. Memory API

man malloc:

```
malloc, free, calloc, realloc - allocate and free dynamic memory

  #include <stdlib.h>

  void *malloc(size_t size); // allocate size bytes, uninitialized

  void free(void *ptr); // release ptr memory

  void *calloc(size_t nmemb, size_t size); // allocate nmemb*size bytes, initialized to zeros

  void *realloc(void *ptr, size_t size); // change allocation to new size
```

Others: strdup(), brk(), mmap()

2. Stack vs. Heap

Stack size is usually limited, heap is not:

```
$ ulimit -a | egrep 'stack|data'
data seg size           (kbytes, -d) unlimited
stack size              (kbytes, -s) 8192
$
```

main program can allocate on stack and pass to functions,
but functions can not allocate on stack and pass back to main:

```
char *f( int n)
{
  char s[n]; // allocated on stack

  return s; // won't work, space for s goes away when function returns
}

char *g( int n)
{
  char *s = malloc(n); // allocated on heap

  return s; // ok
}
```

3. malloc() example

malloc.c:

```
// test malloc() and free()
//
#include <stdio.h>
#include <stdlib.h>

int main( void)
{
  size_t n = 0; char *p, *q = NULL;

  printf( "Enter list of sizes:\n");

  while( scanf("%zi",&n) == 1)
  {
    p = q;
    q = malloc(n); // intentionally allocating new space before freeing old
    free(p);
    printf( "%zi bytes at %p\n", n, (void *) q );
  }

  return 0;
}

/* sample run:

Enter list of sizes:
20
20 bytes at 0x55c6fc852a80
20
20 bytes at 0x55c6fc852aa0
10
10 bytes at 0x55c6fc852a80

*/
```

4. gdb and valgrind examples

If valgrind is not installed: **sudo apt install -y valgrind**

```
$ cat p1-null.c
int main( void)
{
  int *x = 0;

  return *x;
}

$ gcc -g -o p1-null p1-null.c

$ ./p1-null
Segmentation fault (core dumped)

$ gdb -q ./p1-null
Reading symbols from ./p1-null...done.
(gdb) run
Starting program: /home/perry/src/OSTEP/RP/14-vm-api/p1-null

Program received signal SIGSEGV, Segmentation fault.
0x000055555555460a in main () at p1-null.c:5
5         return *x;
(gdb) print x
$1 = (int *) 0x0
(gdb) quit
$ valgrind --leak-check=yes ./p1-null
==28028== Memcheck, a memory error detector
==28028== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==28028== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==28028== Command: ./p1-null
==28028== Invalid read of size 4
==28028==    at 0x10860A: main (p1-null.c:5)
==28028==  Address 0x0 is not stack'd, malloc'd or (recently) free'd
==28028== Process terminating with default action of signal 11 (SIGSEGV)
==28028==  Access not within mapped region at address 0x0
==28028==    at 0x10860A: main (p1-null.c:5)
...
```

5. Other Exercises

Additional exercises from the book:

4. Write a simple program that allocates memory using malloc() but forgets to free it before exiting. What happens when this program runs? Can you use gdb to find any problems with it? How about valgrind (again with the --leak-check=yes flag)?

5. Write a program that creates an array of integers called data of size 100 using malloc; then, set data[100] to zero. What happens when you run this program? What happens when you run this program using valgrind? Is the program correct?

6. Create a program that allocates an array of integers (as above), frees them, and then tries to print the value of one of the elements of the array. Does the program run? What happens when you use valgrind on it?

7. Now pass a funny value to free (e.g., a pointer in the middle of the array you allocated above). What happens? Do you need tools to find this type of problem?