Final review: some of the last topics

COSC 208, Introduction to Computer Systems, 2021-12-10

Announcements

- Final exam
  - Friday noon to 2

Review: memory

- The intended behavior of the program below is to output a string that contains multiple copies of a word (e.g., `"byebye"`). The code below compiles without warnings, but it contains multiple errors.

```
1   #include <stdlib.h>
2   #include <string.h>
3   #include <stdio.h>
4   char *repeat(char *word, int count) {
5       char *dup = malloc(sizeof(*word) * count + 1);
6       int k = 0;
7       for (int i = 0; i < count; i++) {
8           for (int j = 0; j <= strlen(word) * count; j++) {
9               dup[k] = word[j];
10              k++;
11          }
12      }
13      free(dup);
14      return dup;
15  }
16  int main() {
17      char *orig = malloc(4);
18      strcpy(orig, "bye");
19      char *result = repeat(orig, 2);
20      printf("%s\n", result);
21  }
```

For each of the following errors produced by valgrind, describe (in 2-3 sentences) **why** the error is occurring and **how** you would modify the code to correct the error.

- Q1:

```
Invalid write of size 1
    at 0x4006CA: repeat (repeat.c:9)
    by 0x400752: main (repeat.c:19)
Address 0x5204093 is 0 bytes after a block of size 3 alloc'd
    at 0x4C2DB8F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-
linux.so)
    by 0x40066B: repeat (repeat.c:5)
    by 0x400752: main (repeat.c:19)
```

  Not enough space was allocated for the duplicated string: `sizeof(*word)` gets the size of a single character. Use `strlen(word)` to get the number of characters in the original word.

- Q2:

```
Invalid read of size 1
    at 0x4006BF: repeat (repeat.c:9)
    by 0x400752: main (repeat.c:19)
Address 0x5204044 is 0 bytes after a block of size 4 alloc'd
    at 0x4C2DB8F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-
linux.so)
    by 0x400723: main (repeat.c:17)
```

The inner for loop goes beyond the end of the original word. The for loop condition should be `j < strlen(word)`.

- Q3:

```
Invalid read of size 1
    at 0x4E88CD0: vfprintf (vfprintf.c:1632)
    by 0x4E8F8A8: printf (printf.c:33)
    by 0x40076B: main (repeat.c:20)
Address 0x5204090 is 0 bytes inside a block of size 3 free'd
    at 0x4C2EDEB: free (in /usr/lib/valgrind/vgpreload_memcheck-amd64-
linux.so)
    by 0x4006FF: repeat (repeat.c:13)
    by 0x400752: main (repeat.c:19)
Block was alloc'd at
    at 0x4C2DB8F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-
linux.so)
    by 0x40066B: repeat (repeat.c:5)
    by 0x400752: main (repeat.c:19)
```

The string containing the repeated word is free'd (in `repeat`) before it is printed (in `main`). Move the call to `free` in `repeat` to after the call to `printf` in `main`.

- Q4:

```
4 bytes in 1 blocks are definitely lost in loss record 1 of 1
    at 0x4C2DB8F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-
linux.so)
    by 0x400723: main (repeat.c:17)
```

The space for the original word (allocated in `main`) is not freed. Add `free(orig)` after the call to `repeat` in `main`.

Review: threads

- A program contains the following global variables and functions:

```c
void *dbl(void *arg) {
    int *t = (int *)arg;
    *t = *t * 2;
}

void *one(void *arg) {
    int *t = (int *)arg;
    *t = 1;
}
```

*For each of the following main methods, list **all possible outputs** the program could produce. Assume threads are only preempted if they become blocked waiting for other threads.*

- Q5:

```
int main() {
    int *total = malloc(sizeof(int));
    *total = 3;
    pthread_t thrA, thrB;
    pthread_create(&thrA, NULL, &dbl, total);
    pthread_create(&thrB, NULL, &one, total);
    pthread_join(&thrA);
    pthread_join(&thrB);
    printf("%d\n", total);
}
```

  - `1` (if `thrB` runs after `thrA` finishes)
  - `2` (if `thrA` runs after `thrB` finishes)

- Q6:

```
int main() {
    int *total = malloc(sizeof(int));
    *total = 3
    pthread_t thrA, thrB;
    pthread_create(&thrA, NULL, &one, total);
    pthread_join(&thrA);
    pthread_create(&thrB, NULL, &dbl, total);
    pthread_join(&thrB);
    printf("%d\n", total);
}
```

  - `2` (`thrA` is joined, i.e., must finish, before `thrB` is created)

Review: sockets

- Q7: *What sequence of socket functions should a server application call?*
  1. `socket`
  2. `bind`
  3. `listen`
  4. `accept`
  5. `recv`
  6. `send`
  7. `close`
  8. `close`