7/7 Questions Answered

Saved at 2:03 PM

Check-in Quiz 00 C, Memory, fork(), exec()

Q1

1 Point

What are the final values of the following variables by the end of the main program? If a value would be set using "Undefined Behavior" or the program would crash, answer with a question mark (?).

```c
#include <stdio.h>

typedef struct point_st {
  int x, y;
} Point;

typedef struct circle_st {
  int radius;
  Point center;
} Circle;

void mystery(Circle c, int scale, Circle* output) {
  c.center.x *= scale;
  c.center.y *= scale;
  *output = c;
}

int main() {
  Point point = {2, 3};
  Circle circle1 = {3, point} ;
  Circle circle2;

  mystery(circle1, 3, &circle2);
  point.x = 59;
  point.y = 50;

  printf("Circle1 center x: %d, y: %d\n", circle1.center.x, circle1.center.y);
  printf("Circle2 center x: %d, y: %d\n", circle2.center.x, circle2.center.y);
}
```

circle1.center.x

2

circle1.center.y

3

circle2.center.x

6

circle2.center.y

9

Explanation

EVERYTHING is pass-by-value (e.g. passing in a copy) in C. If you see a struct type, it is actually a struct, not secretly a reference to a struct. This includes the Circle struct, which copies the `Point point` when `circle` is initialized.

Save Answer Last saved on **Oct 09 at 2:00 PM**

Q2

2 Points

The following code, when run, has two processes, a child and a parent. The child and the parent each modify and print out a global value.

What are the possible values printed by the parent, and by the child?

If a process can has multiple possible values, please answer with a comma separated list of numbers with no spaces (e.g. 1,2,3)
If there is only one possible answer for a process, please answer with a number only (e.g. 1).

```c
int global_num = 1;

void function() {
  global_num++;
  printf("%d\n", global_num);
}

int main() {
  pid_t id = fork();

  if (id == 0) {
    function();
    return EXIT_SUCCESS;
  }

  global_num += 2;
  printf("%d\n", global_num);
  return EXIT_SUCCESS;
}
```

Parent output:

3

Child output:

2

Explanation

Remember that each process has a separate address space, so changing the value of the global in one process does not change the value in another

Save Answer Last saved on **Oct 09 at 2:01 PM**

Q3

2 Points

How many times does the print statement get executed?

```
int main() {
  for(int i = 0; i < 4; i++) {
    fork();
  }

 printf("a\n");

  return EXIT_SUCCESS;
}
```

please answer with a number only

16

Explanation

The newly forked process continues where its parent left off, and will continue running until it terminates.

Save Answer Last saved on **Oct 09 at 2:01 PM**

Q4 Valgrind Errors

3 Points

In this question, we've taken what we wrote in lecture as get_input.c and slightly modified it so that there is no truncation. This program should work with all inputs <= 100 characters in length.

Here is the modified code with line numbers:

```
 1 #include <unistd.h>  // for read() and write()
 2 #include <stdlib.h>  // for malloc, EXIT_SUCCESS
 3 #include <string.h>  // for strlen
 4
 5 #define MAX_INPUT_SIZE 100
 6
 7 char* read_stdin();
 8
 9 void print_stdout(char* to_print);
10
11 int main(int argc, char** argv) {
12   char* str = read_stdin();
```

```
  if (str == NULL) {
    return EXIT_SUCCESS;
  }

  print_stdout(str);

  return EXIT_SUCCESS;
}

void print_stdout(char* to_print) {

  write(STDOUT_FILENO, to_print, strlen(to_print));
}

char* read_stdin() {
  char* str = (char*) malloc(sizeof(char) * (MAX_INPUT_SIZE + 1));
  if (str == NULL) {
    return NULL;
  }

  ssize_t res = read(STDIN_FILENO, str, MAX_INPUT_SIZE);
  if (res <= 0) {
    free(str);
    return NULL;
  }

  return str;
}
```

Unfortunately, there are some valgrind errors in this code. When run under valgrind and the user inputs "hello" then hits enter, we get this:

```
==1867== Memcheck, a memory error detector
==1867== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==1867== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==1867== Command: ./get_input
==1867==
hello
==1867== Conditional jump or move depends on uninitialised value(s)
==1867==    at 0x484ED28: strlen (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-l
==1867==    by 0x109206: print_stdout (get_input.c:37)
==1867==    by 0x1091E3: main (get_input.c:31)
==1867==
hello
==1867==
==1867== HEAP SUMMARY:
==1867==     in use at exit: 101 bytes in 1 blocks
==1867==   total heap usage: 1 allocs, 0 frees, 101 bytes allocated
==1867==
==1867== LEAK SUMMARY:
```

```
==1867==    definitely lost: 101 bytes in 1 blocks
==1867==    indirectly lost: 0 bytes in 0 blocks
==1867==      possibly lost: 0 bytes in 0 blocks
==1867==    still reachable: 0 bytes in 0 blocks
==1867==         suppressed: 0 bytes in 0 blocks
==1867== Rerun with --leak-check=full to see details of leaked memory
==1867==
==1867== Use --track-origins=yes to see where uninitialised values come from
==1867== For lists of detected and suppressed errors, rerun with: -s
==1867== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

Q4.1 Valgrind Errors: read_stdin
1 Point

One of these errors is accessing unitialized memory:

```
==1867== Conditional jump or move depends on uninitialised value(s)
==1867==    at 0x484ED28: strlen (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-l
==1867==    by 0x109206: print_stdout (get_input.c:24)
==1867==    by 0x1091E3: main (get_input.c:17)
```

Which line should something be added to remove this error? You are not allowed to
function call to something like `memset`, the fix should be a relatively simple operation

Hint: It is not line 24, which is where the access of unitialized memory occurs. Consi
the unitialized memory was accessed and where we can prevent/initialize it.
Hint2: This relates to a really common bug with C "strings" mentioned in lecture

- Line 16
- Line 23
- Line 28
- Line 32
- Line 38

Explanation

The error comes from how we initialize the string, it only later shows up as an iss
when we use it, so the fix is to initialize our string correctly :)

Save Answer Last saved on **Oct 09 at 2:01 PM**

Q4.2 Valgrind Errors: read_stdin fix
1 Point

On the line you selected, what should the line look like after it has been modified?

Note: gradescope can be picky about the formatting of your answer.

- Don't include a semicolon in your answer
- have spaces around any assignment "=" or arithmetic operators "+", "-", etc
  If you are sure you have the right answer but gradescope isn't working, feel free to post privately on Ed and we will confirm

here are some example formats if, if this helps:

`arr[13] = 4 + 6 / 2`

`x = 16 + 2 * arr[0]`

`char c = '\0'`

str[res] = '\0'

Explanation

The problem is that we don't null-terminate the string, `read()` does not put on a null terminator character for you. When we go on to print the string later, it will keep printing characters till it hits the null terminator (which isn't guaranteed to be there unless we add it)

Save Answer Last saved on **Oct 09 at 2:02 PM**

Q4.3 Memory Leak
1 Point

This program also allocates memory with `malloc` but does not free it, causing a memory leak. This memory needs to be `free`'d at some point by calling the free function. Which of the following lines could a call to `free` be added that would resolve the memory leak and have the program behave as expected. If there are multiple answers, choose the line that would be executed first.

Line 16

Line 18

Line 23

Line 38

Explanation

We can't free the allocated memory until it has been used for the last time. Using memory that has already been freed will cause undefined behaviour.

Save Answer Last saved on **Oct 09 at 2:02 PM**

Q5 fork + exec

2 Points

Bellow we have a small program that uses both fork and exec:

```c
#include <stdio.h>   // for printf()
#include <unistd.h>  // for execve()
#include <stdlib.h>  // for exit() and EXIT_FAILURE

int main(int argc, char* argv[]) {
  char* args[] = {"/bin/echo", "T", NULL};
  char* envp[] = { NULL };

  pid_t pid = fork();

  printf("A\n");

  if (pid == 0) {
    execve(args[0], args, envp);
    exit(EXIT_FAILURE);
  }

  printf("O\n");

  return EXIT_FAILURE;
}
```

What is the output of the parent process?

What is the output of the child process?

Do not include new lines in your answer, just the characters such as `TA` or `A` is what is expected

Parent output:

AO

Child output:

AT

Explanation

The child still prints "A" since it comes before the exec and after the fork. The child then prints T from executing `echo T`. The child does not print the O, since once it is done with exec, the process terminates. It does not "return to main" or anything like that. The parent just prints both print statements, and does not exec.

Save Answer Last saved on **Oct 09 at 2:03 PM**

Save All Answers

Submit & View Submission ›