Lecture 10: Heap memory

CSE 29: Systems Programming and Software Tools

Olivia Weng

Announcements

- Problem set 2 due tomorrow at 10am PT
- Initial grades released
- Exam 1

Demo

```
int *reverse(int arr[], int n) {
    int reversed[n];
    for (int i = 0; i < n; i++) {
        reversed[i] = arr[n - i - 1];
    }
    int *to_return = reversed;
    return to_return;
}

int main() {
    int arr[3] = {10, 20, 30};
    int *reversed = reverse(arr, 3);
    for (int i = 0; i < 3; i++) {
        printf("%d\n", reversed[i]);
    }
}
```

What is the heap for?

Heap memory

- Heap memory can be accessed from any function as long as we have pointers to it

- What if we want to create an array that persists after a function returns?
  - Use the heap!

How to request memory from the heap?

- `malloc(num_bytes)`
  - stands for "memory allocate"
  - returns the address of the first byte in the memory allocated
  - part of stdlib.h
- Example: variable-length array (length of the array changes over time)

```
int *pa = malloc(3 * sizeof(int));

// can do whatever array things I want with pa
```

Demo

- `reverse_fix()`

reverse_fix()

```
int *reverse_fix(int arr[], int n) {
    int *reversed = malloc(n * sizeof(int));
    if (reversed == NULL) {
        printf("Memory allocation failed!\n");
        return NULL;
    }
    for (int i = 0; i < n; i++) {
        reversed[i] = arr[n - i - 1];
    }
    return reversed;
}
```

What happens to heap memory after `malloc()`?

- Heap memory stays there forever so other functions can use it
  - Nice, but what happens when you're done?

We must `free()` the heap

- `free(void *ptr)`
  - Tells the heap to free memory that it allocated at this pointer
  - Now other processes can use it

- What happens if we do not free()?
  - Memory leak!
  - Performance degradation (slow down the computer)
  - Other processes cannot get the memory they need

Demo

- valgrind

With heap memory, comes great responsibility

- You are responsible for the memory you request from the heap
  - Manual memory management
  - Each call to `malloc()` should have a corresponding call to `free()` to prevent memory leaks

- What if I call `free()` on memory that has already been freed?
  - Usually segfault (depends on the system)!
- What if I call free() on memory that was never malloc'd?
  - segfault!

Memory APIs

- `malloc()`: allocate memory on the heap

```
int *pa = malloc(10 * sizeof(int));
```

- `free()`: free allocated memory

```
free(pa);
```

- `calloc()`: similar to malloc but zeros out allocated memory

```
int *pa = calloc(10, sizeof(int));
```

- `realloc()`: increase or decrease size of an allocation
  - grows or shrinks the heap allocation or copies the data to a new allocation if needed

```
pa = realloc(pa, 20 * sizeof(int));
```

Creating structures on the heap

```
struct point *p = malloc(...);

p->x = 3;

p->y = 4;

// … do something with p

free(p); // important!
```

Creating structures on the heap

```
struct point *p = malloc(sizeof(struct point));

p->x = 3;

p->y = 4;

// … do something with p

free(p); // important!
```

Demo

- Strings in python

How could we implement a String class in C?

How could we implement a String class in C?

```
struct string {
    uint64_t length; // = strlen(contents)
    char *contents; // has space for length + null terminator
};

typedef struct string String;
```