Swapping

CSC 213 – February 15, 2021
Agenda for Today

1. Bug of the Day
2. Assignment and Lab Q&A
3. New Assignment: Queue
4. Swapping
5. Wrap Up
Bug of the Day
What's wrong here?

1. `int* x = malloc(2*sizeof(int));`
2. `x[1] = 2;`
3. `x[2] = 3;` ← this seems bad!
4. `free(x);`

This is a buffer overflow. We may see a segfault, or the write might be "fine." The second case is much more likely, but it's still bad.
Do we need to read in data in our print_contents function?
   A: No, the data parameter points to the beginning of the file contents, which have been fully read before print_contents is called. The size parameter tells you how many bytes are in the file.

How bad is it if we don’t handle odd-length files correctly?
   A: Not terrible, but you may lose a few points (not more than 10). A reasonable solution to this problem could be just one or two lines of code.

How should we handle the case where the file has the wrong signature?
   A: I believe the code that reads the file in checks the signature, so you can assume it’s correct. You can also assume I’m not going to give you corrupted files (e.g. the file ends before one of its file entries is finished).

Can files contain newlines?
   A: Yes, but you shouldn’t have to do anything special for them.
What problem do we solve by using structs for the file header?

A: One of the tricky parts of this assignment is keeping track of offsets in the file contents, and where each field you want is located in that array. With a struct, you can let the compiler do some of that work for you. If you interpret a point in the input file as the beginning of a struct, you can access fields in that struct without worrying about their offsets.

```c
void print_contents(uint8_t* data, size_t size) {
    header_t* header = (header_t*) &data[8];
    // Now it's okay to do this:
    for (int i=0; i<16; i++) {
        printf("%c", header->identifier[i]);
    }
    // TODO: Make sure header->size has a null terminator at the end of it
    int entry_size = atoi(header->size);
    ...
}
```
Archive Printer Assignment Q&A

Why does the archive format like to align headers to even offsets?

A: This a format that has to work across a lot of different kinds of computers. Some computers do not like to make unaligned accesses, meaning loads from pointers that have odd values, values that aren’t multiples of 4, 8, etc. This format is designed so you’ll never have to load headers from an odd address.

Is the best way to read content from a struct using for loops, or should we be using memcpy and friends?

A: It’s really up to you. In some situations (not necessarily this one) it may be much better to use the data in place instead of copying it somewhere. You might not use memcpy in that situation, but sometimes you have to.
Virtual Memory Lab Q&A

How do we compare info->si_addr to the addresses of the chunks?
A: They should both be eight bytes. If you print an eight-byte pointer with the %p printf flag, it looks like this: 0x7fff11112222 (up to 16 characters, but printf drops leading zeros). I guarantee sizeof(void*), sizeof(intptr_t), sizeof(info->si_addr), etc. are all eight bytes.

In chunk_copy_lazy there are numbered steps, then some instructions marked “later”. When do we do the “later” steps?
A: They’ll happen in the segfault signal handler.

How do we find out which chunk was written, causing a segfault?
A: The info parameter in the segfault handler tells you about where the signal came from. One field, info->si_addr, holds the address of the access that caused the segfault. If you cast that to an intptr_t, you can compare it to the start and end addresses of the chunks you’ve created. Presumably you’ve stored those somewhere.
Virtual Memory Lab Q&A

$p_1 \rightarrow A \leftarrow$ mapped to some physical memory (read-only)

$p_2 \leftarrow \text{ lazy copy of } p_1$

$p_2 \rightarrow B \leftarrow$ mapped at the same address chunk B was at.

$C \leftarrow$ segfault here

1. copy from $B$ to $D$
2. use mmap to replace $B$ with $C$
3. copy from $D$ to $C$

$D$ can be a local, or on the heap - it's up to you.
Virtual Memory Lab Q&A

Do we need to keep track of both virtual and physical addresses for chunks?
A: You cannot, as a user, access information about physical memory. All you can keep track of is virtual memory. You have to keep track of which things are mapped to the same physical chunks, and which ones are separate mappings.

What should the program do when it works?
A: There are two measurements: the time spent copying, and the time spent writing to some subset of those copies. Your lazy copying time should be about 10% of the eager copying time. And the lazy writing time will be higher than eager writing time. These numbers vary a LOT on repl.it, so don’t worry if yours don’t match exactly.

Can we use global variables to keep track of where chunks are located?
A: Yes, you should. These are fundamentally global state, so it would be weird (and difficult) to implement them using locals. Try not to use too many; something like a linked list head pointer could be your only global variable.
Assignment: Queue
Swapping
Memory Limits

What happens when a process runs out of virtual address space?

**It will just crash:** there’s no more way to give the program more virtual address space. The program may need space for stack, heap, globals. What can a program do besides crash if these aren’t available?

**Possibly less catastrophic:** malloc() may return NULL instead of a useful pointer

What can we do to fix it?

**Check if the program is running out of memory in advance:** if you discover you are running out of space, you could report and error and exit. You could also free some things you don’t need to make more space.

**Just make sure we never run out.** A 64-bit computer uses 64-bit values as virtual addresses. There are $2^{64}$ possible virtual addresses available, which is really really really really really huge.
Memory Limits

What happens when a computer runs out of physical memory?

**The program will stop:** we could give the program some virtual addresses, but there’s no physical memory to connect those to.

What can we do to fix it?

**Make space in physical memory by “evicting” pages.** The OS can disconnect a process’ virtual addresses from physical memory, and then use that physical memory for another process, or another mapping in the same process.

In order to preserve the contents of that reclaimed memory, we have to save it somewhere. We’re out of physical memory, so we’ll have to save it on a hard drive. Eventually, the program may access the evicted memory, which we’ll restore from disk by evicting something else.
Reminders

**Archive Printer Assignment**
Due tonight at 11:59pm

**Virtual Memory Lab**
Due tomorrow at 11:59pm

**Read the lab before class tomorrow**
I am making some revisions, but will have it posted by this afternoon.

Also, read the assigned reading!