CSE 333: Systems Programming

Section 2

Memory ownership and GDB

Section format changes

- I'll go over some material, ask questions along the way, and then you'll do an exercise in teams of two
- Turn in solution file(s) to Dropbox for the section by 11:59pm of the day of section
  - One submission per group; leave a comment with your partner's name
  - It should be possible to finish everything in section
- If you miss a section, upload your code/other files to the Dropbox and email me the answers to the questions asked in the section slides

Memory ownership

- Good design practice: identify which agents own which heap-allocated memory
- Agents that own heap-allocated memory are responsible for freeing it unless they transfer ownership
- Ownership and ownership transfer should be explicit
  - What can we do in to make memory ownership and ownership transfer explicit or at least more obvious?

Memory ownership

- Example: I want to write a function that processes some work and returns whether all processing succeeds
- doWork should somehow surface an error through the error parameter if it fails

```
bool doWork(WorkItem* work_items,
            int num_items, ?? error, ...);
```

Memory ownership

```
bool doWork(WorkItem* work_items,
            int num_items, ?? error, ...);
```

- Some possibilities:
  - doWork heap-allocates an error string and returns a pointer to it in *error (i.e. make error of type char**)
  - The caller heap- or stack-allocates an error string and passes it to doWork (i.e. make error of type char* and pass its length as well)
- What are the tradeoffs between these approaches with respect to memory management? Any other possibilities?

Using GDB

- GDB is the Swiss army knife of debugging
- GDB lets you examine the state of a running program, watch its behavior, and even modify its state
- Basic usage:

```
$ gdb ./program-name
gdb) start
... (set breakpoints)
gdb) continue
```

Using GDB

- Use the “p” (print) command within GDB to print out values of variables and their addresses
- Use the “b” (breakpoint) command to set a breakpoint at a particular line/file, e.g. “b 79” to break execution at line 79 in the current file
- Use the “c” (continue) command to resume execution after hitting a breakpoint
- Use the “d” (delete breakpoint) command to remove breakpoints, e.g. “d 1” to delete breakpoint 1
- Use the “list” command to output the code with line numbers in the current file. “list [line-#]” will list code from the given line; press Enter to see more code