COSC 3750

More of The Shell

Kim Buckner

University of Wyoming

Apr. 13, 2021

What it does

- Your shell will initialize whatever is needed.
- It will then start processing the input.
- You already know how to do this, a line at a time.
- The parser I gave you will handle the lines of input.

What you <u>should</u> be doing

- Make a **real plan** for the whole shell.
- Implement Part I as though you really intend to finish the job.
- I suggest some method of maintaining state.
- Any ideas how to do this?

Structures and suggestions

```
typedef struct node Node;
typedef struct word Word;

struct node {
  struct node *next,*prev;
  char *command;
  Word *arg_list;
  int input, output, error;
  char *in_file, *out_file, *err_file;
}
```

back

(more . . .)

```
struct word {
  struct word *next,*prev;
  char *string;
}
```

(more . . .)

- I would declare a couple of pointers, “Node *Head, *current;”, in main() and set them to NULL.
- When I start processing a **line** of input, check to see if Head is NULL.
- If it is, use `calloc()` to create a new Node.

```
Head=calloc(1,sizeof(Node));
current=head;
```

(more . . .)

- Use that Node to record the information about the first command.
- Use the “current” pointer ALWAYS, NOT Head.
- I would use the three ints in the struct for the the file descriptors, STDIN_FILENO and so forth.

(more . . .)

- The “arg_list” is a simple linked list of Words that can store any number of arguments.
- It is easy to write functions (to keep the main code clean) to handle everything.
- Like “void insertWord(Word *list, const char* arg)”.

(more . . .)

- The child processes will NOT have to "clean" anything up (free memory) because they will either exit if `exec()` failed, or they will no longer exist.
- But the main shell cannot afford to have a bunch allocated memory that is leaked.
- So write a function like "clean_up(Node *node)" that will traverse and free all the allocated memory.

(more . . .)

- ALWAYS REMEMBER, if you use **calloc()**, **malloc** or any of the others, check the return value for NULL.

  ```
  Head=calloc(1,sizeof(Node));
  if(Head == NULL) {
    perror("calloc()");
    return 1;
  }
  ```

- If any of this is confusing get a hold of me and we should be able to straighten it out.

Memory allocation

- Why use ***calloc()***?

Memory allocation

- Why use ***calloc()***?
- Because it not only allocates the memory, it “clears” it.
- It sets all the bytes allocated to 0.
- And why is that a good thing?

(more . . .)

- Look at the struct node.
- What jumps out at you?

(more . . .)

- Look at the struct node.
- What jumps out at you?
- There are what, seven pointers?
- Fortunately, the compiler is happy comparing 0 with any pointer, especially NULL.

(more . . .)

- Which all means that you only have to initialize the integers used to indicate what is happening with the standard files.
- Nice and easy.

What you will probably do

- Some, I hope not many, of you will start keeping a BUNCH of “flag” variables to keep track of things.
- That is okay for a couple but it does NOT grow with the program.
- The more complex the program, the more flags you have, the more checks of the flags you make, the more errors you introduce.

PIPE

- Pipe is just a special version of redirection.
  - the output of the left hand is redirected to ...
  - the input of the right hand side is redirected from ...
- The other side of the pipe.

Processing

- When we see a pipe there is some state to verify.
- First there must be a "current" command.
- Then the current command cannot have its output already redirected.
- What do we do with the file name field? Leave it NULL.

(more . . .)

- That way we can look at the ”output” variable, see it is PIPE and ignore the filename.

Then

- That takes care of the current command.
- What about the NEXT command?
- First, we have not even seen it yet.
- Second, how then do we redirect its input?

Because

- When we see the pipe, must assume that the current command is done.
- Add the current command to the list of commands.
- Create a new current command
- Add the redirection to the new current command.
- Set the input field to PIPE.

(more . . .)

It is a linked list after all so

```
current->next=calloc(1, sizeof(Node));
current=current->next;
current->input=PIPE;
```

And

- If the command is NULL, the PIPE must be redirecting the input of the NEXT command.
- Which means that we have to create an empty state structure and fill in the information that its input will be redirected.
- Now, suppose there is no next “word?” What will you do?

Summary

- Write a program with an endless loop
- Process the user input a line at a time
- For every line
  - Create a set of objects
  - Use a set of your functions that add elements to objects
  - These validate the “state” as they modify the objects.
  - Once at the end of the line, print the data from the set of objects.