Heap manager review and intro to shell

COMPSCI210 Recitation

28 Jan 2013

Vamsi Thummala

Heap manager: review

- What's metadata?
  - Data about data
  - How is it useful in the heap manager?
- Memory alignment
  - ALIGN Macro
  - New C Standard
    - C11: void *aligned_alloc(size_t algn, size_t size);
- Pointer arithmetic and casting
  - int *ptr = dmalloc(1)
  - int *next = (void *) ptr + 1
- Pointer manipulation
  - Infinite loop
    - ptr->next = ptr
  - segfault issues
- Space utilization (success rate)
- Time complexity

The facts

Java to C: Pointers are evil!

No one shot solution: Lot of design choices and tradeoffs

Debugging segfaults is hard!

gdb can help

Code walk through is often faster (for this lab)

Designing the data structure

- How do we know where the chunks are?
- How do we know how big the chunks are?
- How do we know which chunks are free?
- Remember: no queuing of buffer calls to malloc and free… must deal with them real-time.
- Remember: calls to free only takes a pointer, not a pointer and a size.
- Solution: **Need a data structure to store information on the “chunks”**
- Where do I keep this data structure?

Data structure requirements

- The data structure needs to tell us where the chunks are, how big they are, and whether they’re free
- We need to be able to CHANGE the data structure during calls to malloc and free
- We need to be able to find the **next free chunk** that is “a good fit for” a given payload
- We need to be able to quickly mark a chunk as free/allocated
- We need to be able to detect when we’re out of chunks.
  - What do we do when we’re out of chunks?

No external space

It would be convenient if it worked like:

```
malloc_struct malloc_data_structure;
…
ptr = malloc(100, &malloc_data_structure);
…
free(ptr, &malloc_data_structure);
…
```

Instead all we have is the memory we are giving out.

All of it does not have to be payload! We can use some of that for our data structure.

The data structure

The data structure IS your memory!

A start:

<h1> <ptr1> <h2> <ptr2> <h3> <ptr3>

What goes in the header?

- That’s your job!

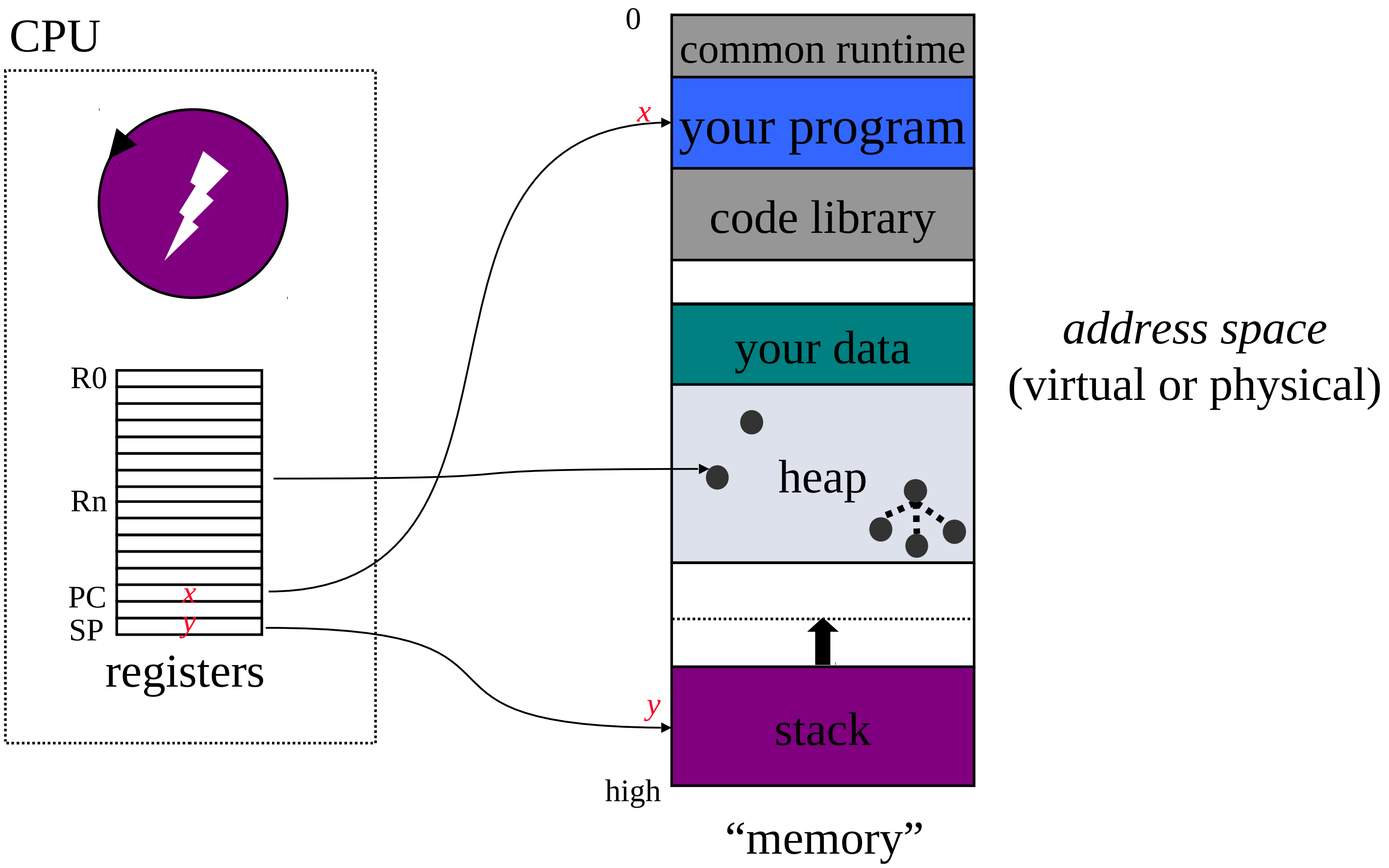
Lets say somebody calls free(ptr2), how can I coalesce?

- Maybe you need a **footer**? Maybe not?

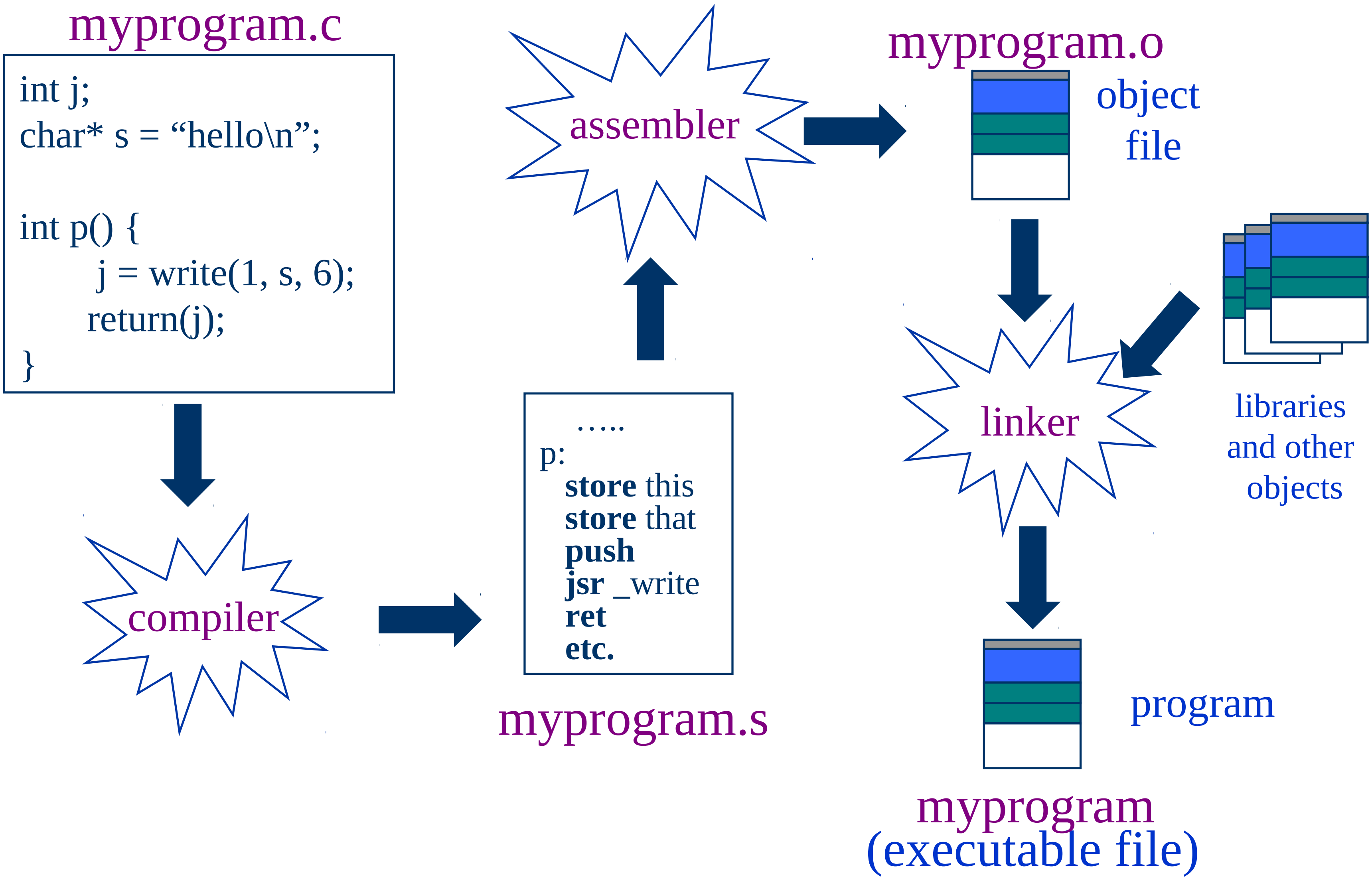
Design considerations

- Free blocks: address-ordered or LIFO
- What’s the difference?
- Pros and cons?
- What are the efficiency tradeoffs?
- Heap vs. List

Heap manager: A larger context



The Birth of a Program (C/Ux)



myprogram.c

```
int j;
char* s = “hello\n”;

int p() {
        j = write(1, s, 6);
    return(j);
}
```

compiler

```
    …..
p:
  store this
  store that
  push
  jsr _write
  ret
  etc.
```

myprogram.s

assembler

myprogram.o

object
file

linker

libraries
and other
objects

program

myprogram
(executable file)

A quick reminder

- Heap manager is due today!
- Submission guidelines
- Policy on cheating

Next Lab: A Devil Shell (dsh)

Shell

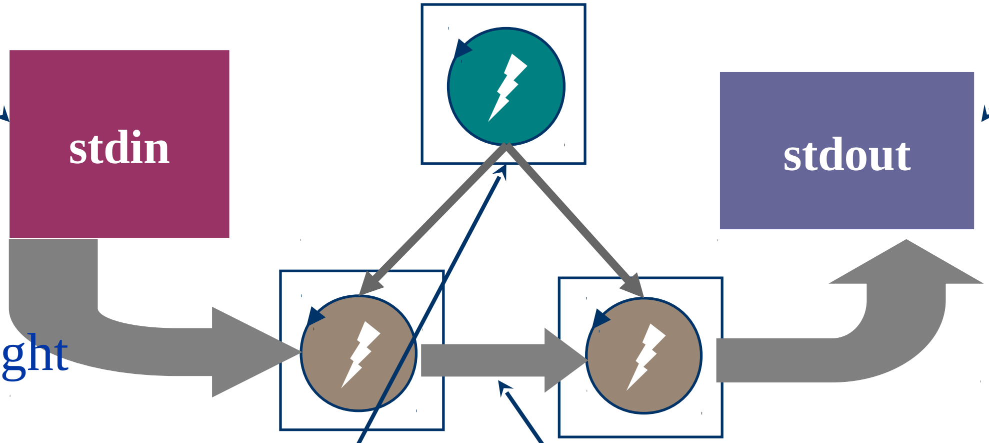
- Interactive command interpreter
- A high level language (scripting)
- Interface to the OS
- Provides support for key OS ideas
    - Isolation
    - Concurrency
    - Communication
    - Synchronization

Demo

Unix programming environment

Standard unix programs read a byte stream from standard input (fd==0).

They write their output to standard output (fd==1).



Stdin or stdout might be bound to a file, pipe, device, or network socket.

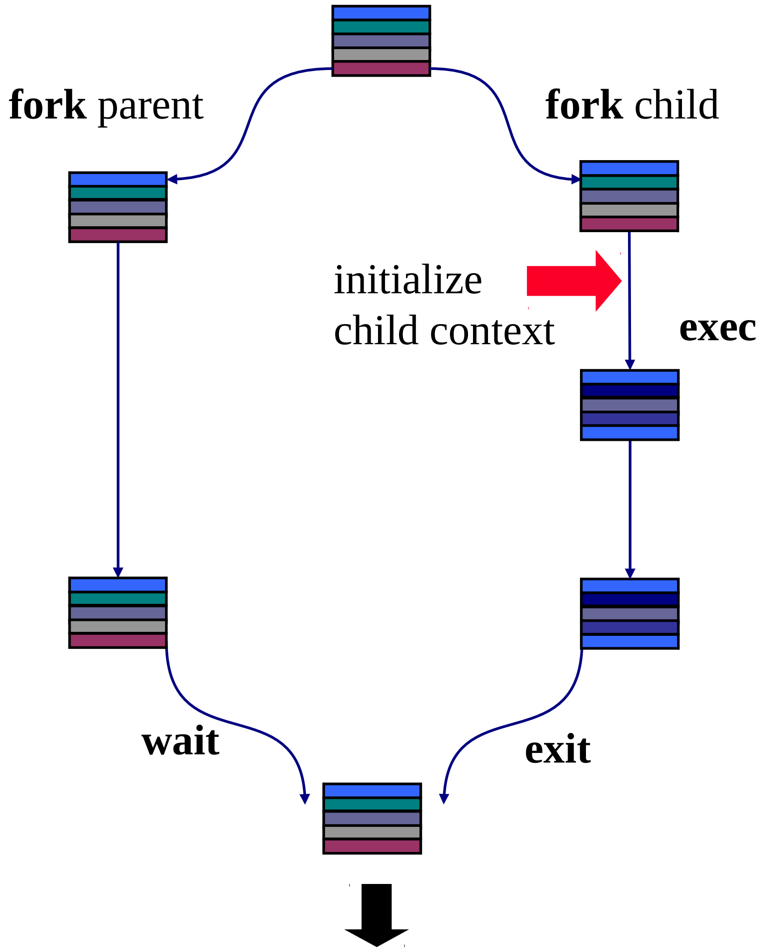
If the parent sets it up, the program doesn’t even have to know.

That style makes it easy to combine simple programs using pipes or files.

Shell Concepts

- Process creation
- Execution
- Input/Output redirection
- Pipelines
- Job control
  - Process groups
  - Sessions
  - Foreground/background jobs
    - Given that many processes can be executed concurrently, which processes should have accesses to the keyboard/screen (I/O)?
  - Signals
    - SIGTTOU, SIGTTIN, SIGINT, SIGCONT, SIGSTP

Unix fork/exec/exit/wait syscalls



int pid = fork();
Create a new process that is a clone of its parent.

exec*("program" [, argvp, envp]);
Overlay the calling process with a new program, and transfer control to it.

exit(status);
Exit with status, destroying the process. Note: this is not the only way for a process to exit!

int pid = wait*(&status);
Wait for exit (or other status change) of a child, and "reap" its exit status. Note: child may have exited before parent calls wait!

Process creation and execution

```
while (1) {
 printf(“dsh$ ”);
   command = readcmdline(args);
   switch (pid = fork()) {           // new process; concurrency
     case -1:
         perror(“Failed to fork\n”);
     case 0:                                // child when pid = 0
         exec (command, args, 0);          // run command
      default:                             // parent pid > 0
         waitpid(pid, NULL, 0); // wait until child is done
 }
```