15-213 Recitation 7 – 3/5/01

Outline

- Control Flow
- Memory Allocation
  - Lab 3 Details

Reminders

- Lab 3: Conservative Garbage Collector
  - Checkpoint Due 3/13
  - Lab Due 3/21
- At least there's nothing over Spring Break (I think)

Shaheen Gandhi

e-mail:

sgandhi@andrew.cmu.edu

Office Hours:

Wednesday 1:30 – 2:30

Wean 3108

Exceptional Control Flow

- Higher level abstractions for dealing with miscellaneous conditions:
  - Error conditions that require errors to be thrown up many stack frames (functions)
  - Interrupt handling (I/O – Keyboard, Mouse, Network, etc.)
  - Familiarize yourself with `wait(2)`, `exec(3)`, `fork(2)`, `signal(2)`, etc.

`sigsetjmp(3)` & `siglongjmp(3)`

`sigsetjmp(3)`

- Saves state about:
  - Stack Context
  - Registers
  - Program Counter
  - Blocked Signals

`siglongjmp(3)`

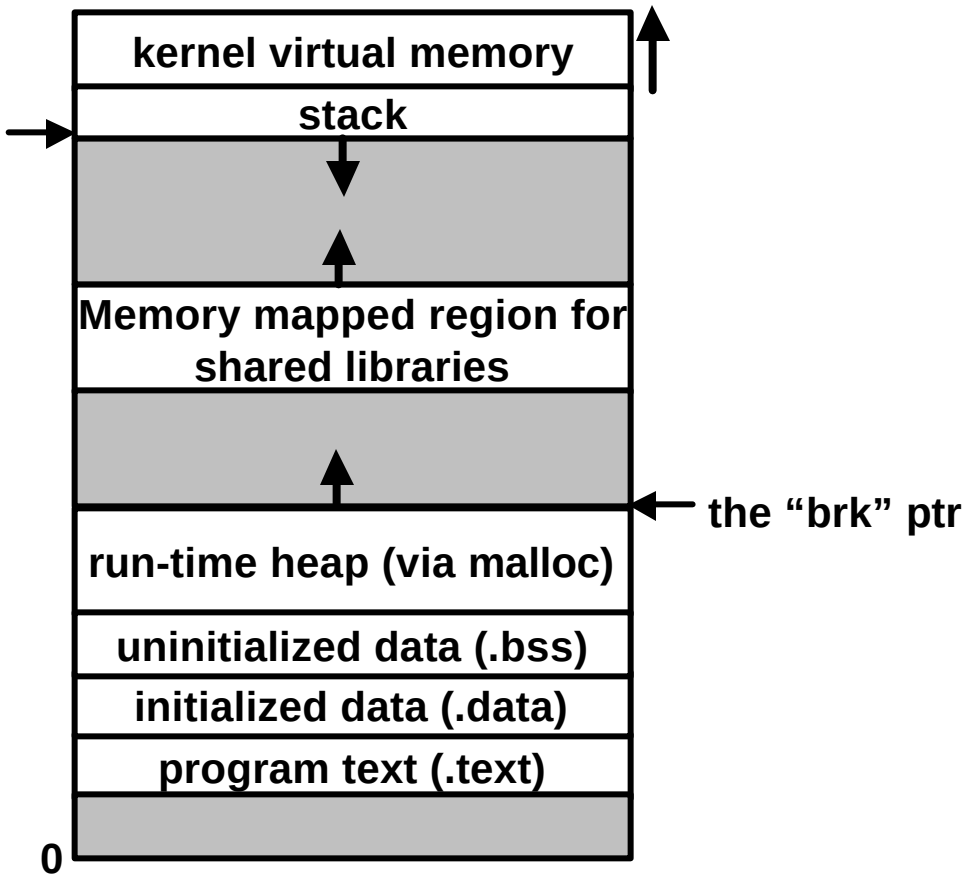
- Starts executing by immediately executing code from `sigsetjmp()`
- Man page says: “setjmp() and sigsetjmp make programs hard to understand and maintain. If possible an alternative should be used.”
  - But we make you do it anyway

Dynamic Memory Allocation

- Applications need variable amounts of memory (unknown at compile time).
- Use dynamic memory allocation to reserve chunks of memory at run-time.
- Equivalent to the `new` operator in Java is `malloc(3)` in C
- `free(3)` un-reserves ('frees') `malloc`'d memory.
  - No equivalent in Java, since Java does nifty garbage collection

Dynamic Memory Allocation: How it's done

First a picture



Dynamic Memory Allocation: How it's done

- The `malloc` package maintains the state of the run-time heap
  - Basically tons of grungy pointer arithmetic
  - The "Heap" is just the area between two addresses: `dseg_lo` and `dseg_hi`
- `malloc` must find a valid contiguous block of memory in the heap and return this to the application
- `free` must return the unused space to the heap for further allocations
- How the functions do this is entirely up to you
  - Generally, fast and inefficient is preferrable to complex, efficient designs, although both are primary concerns

Conservative Garbage Collection

- Reclaim unused space from the application
  - So we can use it to fill future allocations
- How to do it?
  - Mark and Sweep
    - Find everything you (might) need
    - Reclaim the rest
- Where do you find everything you need?
  - Start with the “roots”
    - Current registers
    - Stack
    - Heap (yes, the same one `malloc` manages)
  - Then do a depth first search on the data you find

Conservative Garbage Collection: An Example

```
typedef struct node
{
 struct node *next;
} Node;

void taste(Node *h)
{
 Node *n = h;
 h = malloc(sizeof(Node));
 h->next = n;
}

void srees(Node **h)
{
 Node *n = *h;
 *h = (*h)->next;
 free(*h);
}
```

```
Node *head = malloc(sizeof(Node));
head->next = malloc(sizeof(Node));
taste(head);
srees(&head);
```

What's wrong?

Lab 3 Tips

- Start Early
  - Checkpoint next Wednesday
  - You need a working collector by then
- Read the Lab handout
  - Now read it again
- Don’t write any code until you know what you want to do
- Review Pointer Arithmetic