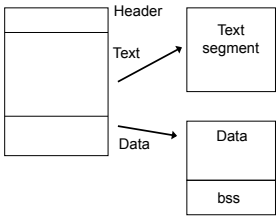
How does trap really work?

August 28, 2018

Y. Charlie Hu

Once upon a time, life was simple

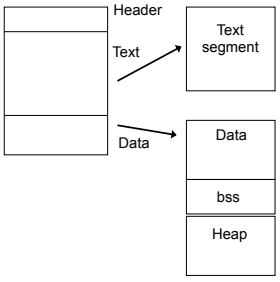
```
int sum;
int a=3;
Int b=4;
int main() {
     sum = a + b,
    printf("%d\n", sum);
}
```



2

Adding dynamic data alloc

```
int sum;
int a=3;
Int *b;
int main() {
      b = malloc(4); *b =4;
      sum = a + b,
     printf("%d\n", sum);
     free(b);
}
```

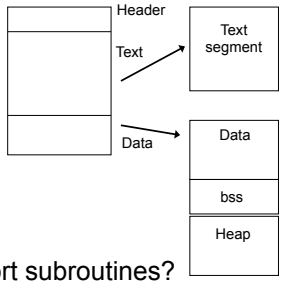


3

Adding subroutines

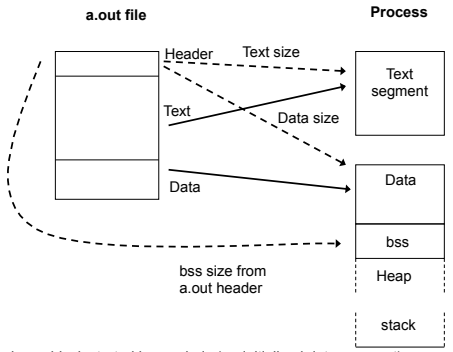
```
int main(){
     int sum;
     sum = add(3, 4);
      printf("%d\n", sum);
}

int add(int a, int b){
   int sum;
   sum = a+b;
   return sum;
}
```



How to support subroutines?

Loading



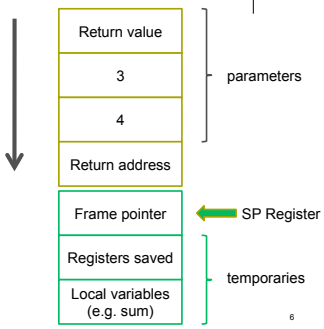
bss = block started by symbols (un-initialized data segment)

5

Adding subroutines

```
int main(){
    int sum;
    sum = add(3, 4);
    printf("%d\n", sum);
}

int add(int a, int b){
   int sum;
   sum = a+b;
   return sum;
}
```



6

Who generates code to manage stack for subroutine calls?

7

x86 C Calling Convention

- A calling convention is an agreement among software designers (e.g. of compilers, compiler libraries, assembly language programmers) on how to use registers and memory in subroutine
  -- There exist many calling conventions
- NOT enforced by hardware
- Allows software pieces to interact compatibly, e.g. a C function can call an ASM function, and vice versa

8

x86 C Calling Convention

- Questions answered by a calling convention:
  - How to preserve old context and create new Context?
  - How to pass parameters?
  - How to save CPU register values?
  - How to store local variables?
  - How to return values?

9

System call (DLXOS)

- Just like a subroutine call, except
  - It executes “trap” instruction
  - “trap” instruction transfers control to _intrhandler (callee) in dlxos.s (by hardware)
  - _intrhandler
    - Saves caller context (registers etc)
    - Calls dointerrpt in traps.c
  - dointerrupt salls intrreturn
  - _intrreturn n dlxos.s
    - Pops stack back off

10