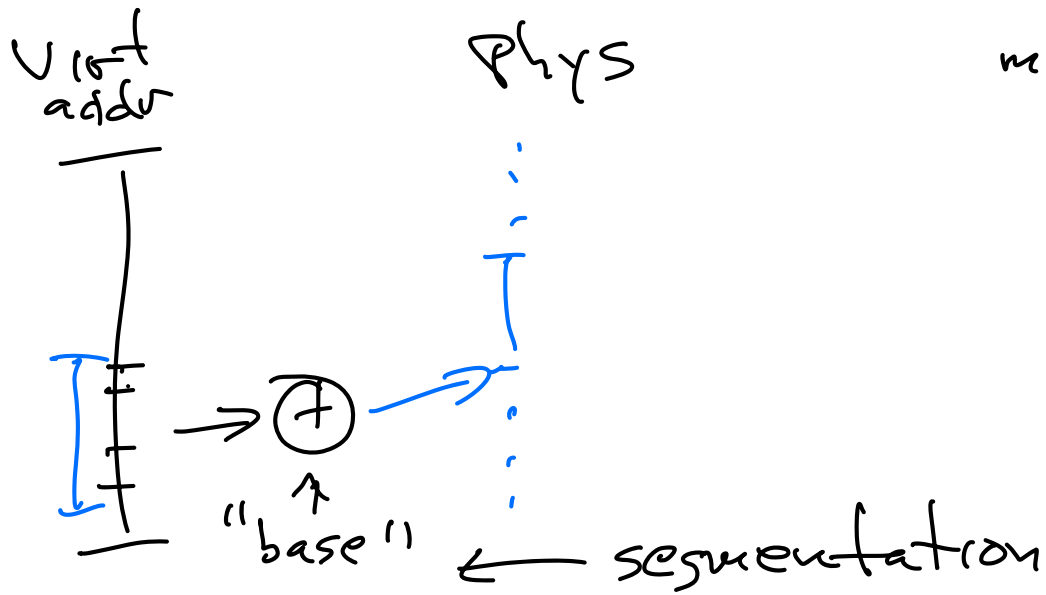
CS 3650 – Computer Systems
Spring 2024
Peter Desnoyers

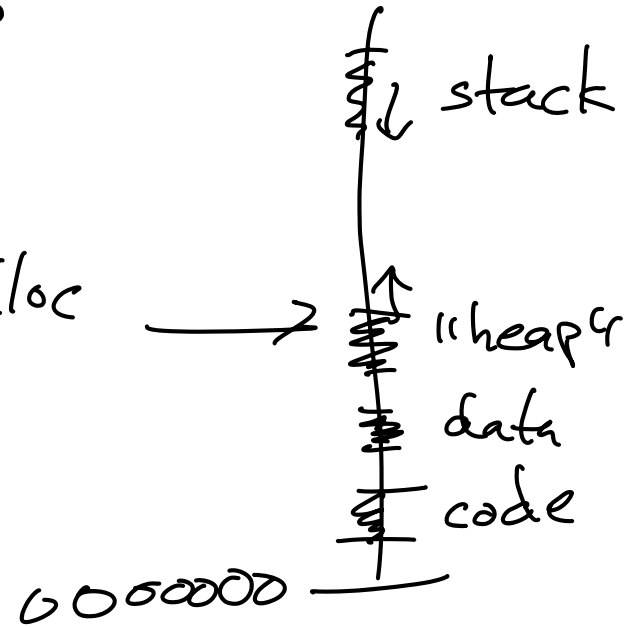
Lecture 9, Tue Feb 6 2024

multiple redirects - ls > file1 >file2

process memory space
virtual memory



process



Executable format & header

mov 4 → eax — sys_write

1 → ebx

"hello world\n" → ecx

12 → edx

syscall

trap → INT x80

system call ABI

↑

binary

syscall # → EAX

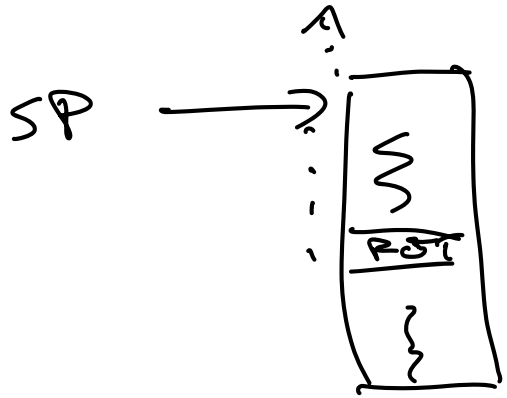
arg 1 → EBX

..2,3 → ECX, EDX

write(1, "hello world\n", 12)

12

x86. registers: EAX EBX ECX EDX (a few others)



→ _start

main( )

printf( )

RET

---

gcc -static test.c

→ self-contained executable

_startup:

various init code

main

foo

printf

write

...

How a program uses the stack

→ arguments (compiler)

→ return addr (CALL instruction) ← but not for ARM...

→ local variables

≈≈≈

CALL: PUSH PC+<1 instr>

ADD

SUB

...

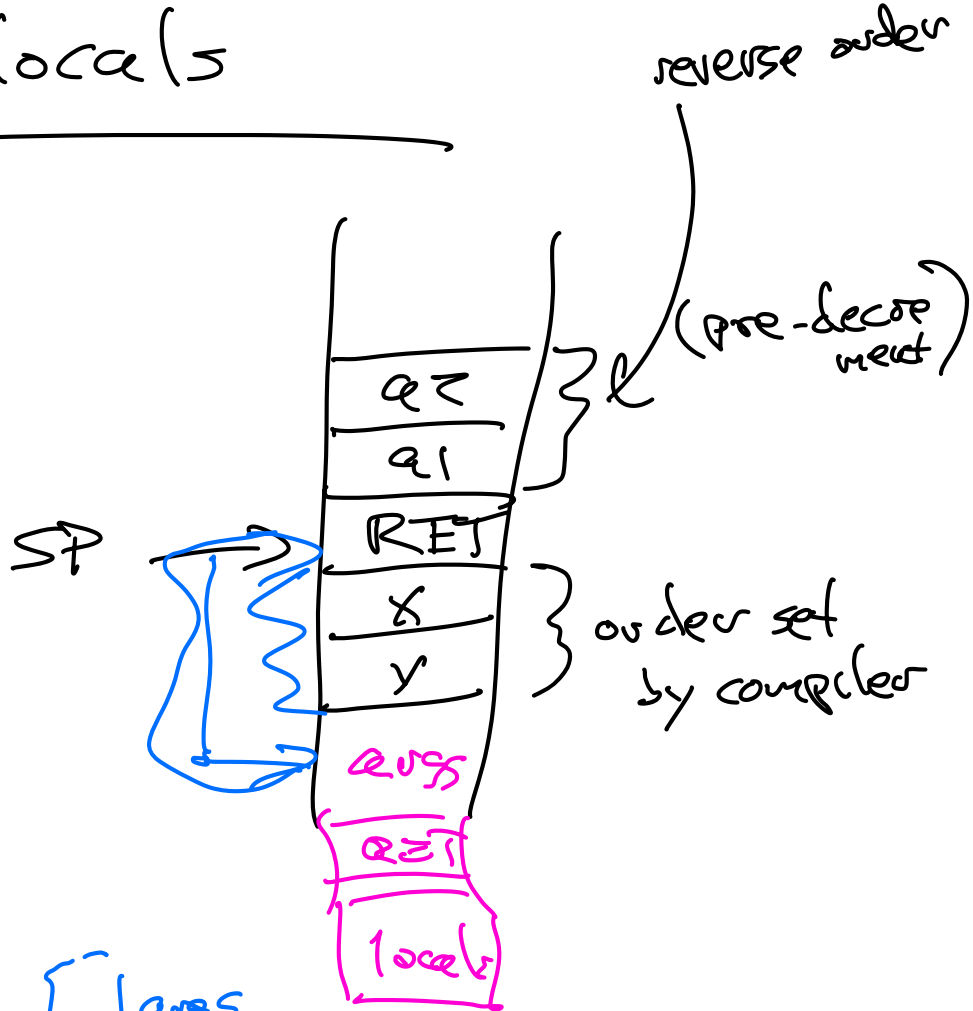
RET Jump to <POP>

```
foo(1,2)
:
bar(3,4)
        printf(arg1,5)

foo(int a1, int a2){
    int x,y;
    bar(a1) →
```

Arguments & locals

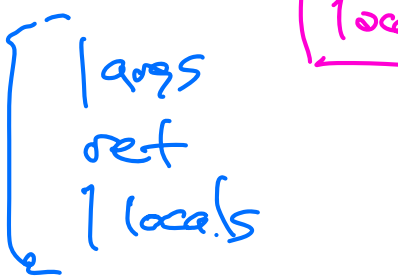
```
foo(int a1, int a2){
    int x, y
```



assembly code:

```
SP = SP - sizeof(locals)
```

```
SP = SP + sizeof(locals)
RET
```



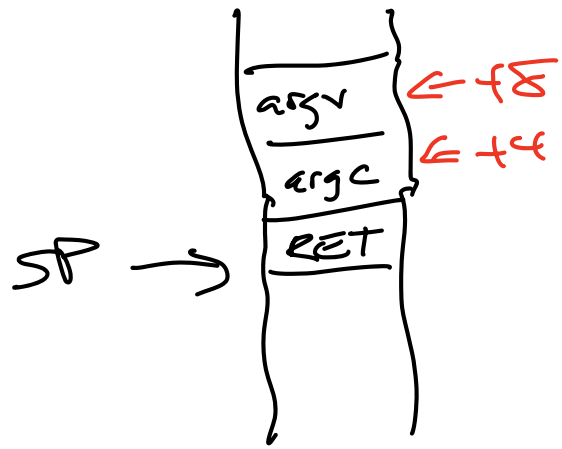
main (int argc, char ** argv)

arg passing:

approach 1: push on stack

2: registers

e.g.
syscall example



LOAD R1 ← *(SP+4)

argc

LOAD R2 ← *(SP+8)

argv

Frame pointer / base pointer

-fno-omit-frame-pointer ← tell optimizer to leave stack info for debugger

function:
  push FP
  FP = SP
  :

  SP = FP
  pop FP
  RET

what's wrong with this:

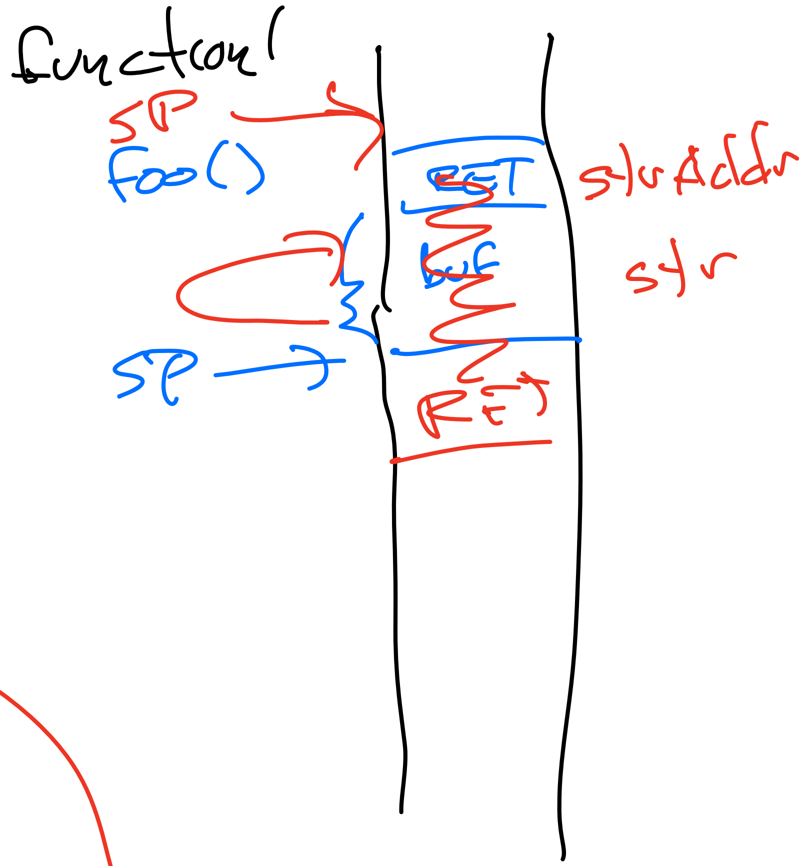
```
char * foo () {
    char buf[] = {'a', 'b', 'c', 0};
    return buf
}
```

what's wrong with this?

```
char * foo() {
    char buf[] = {'a', 'b', 'c', 0};
    return &buf
}
```

```
function1() {
    char * str = foo()
    printf("%s\n", str);
}
```

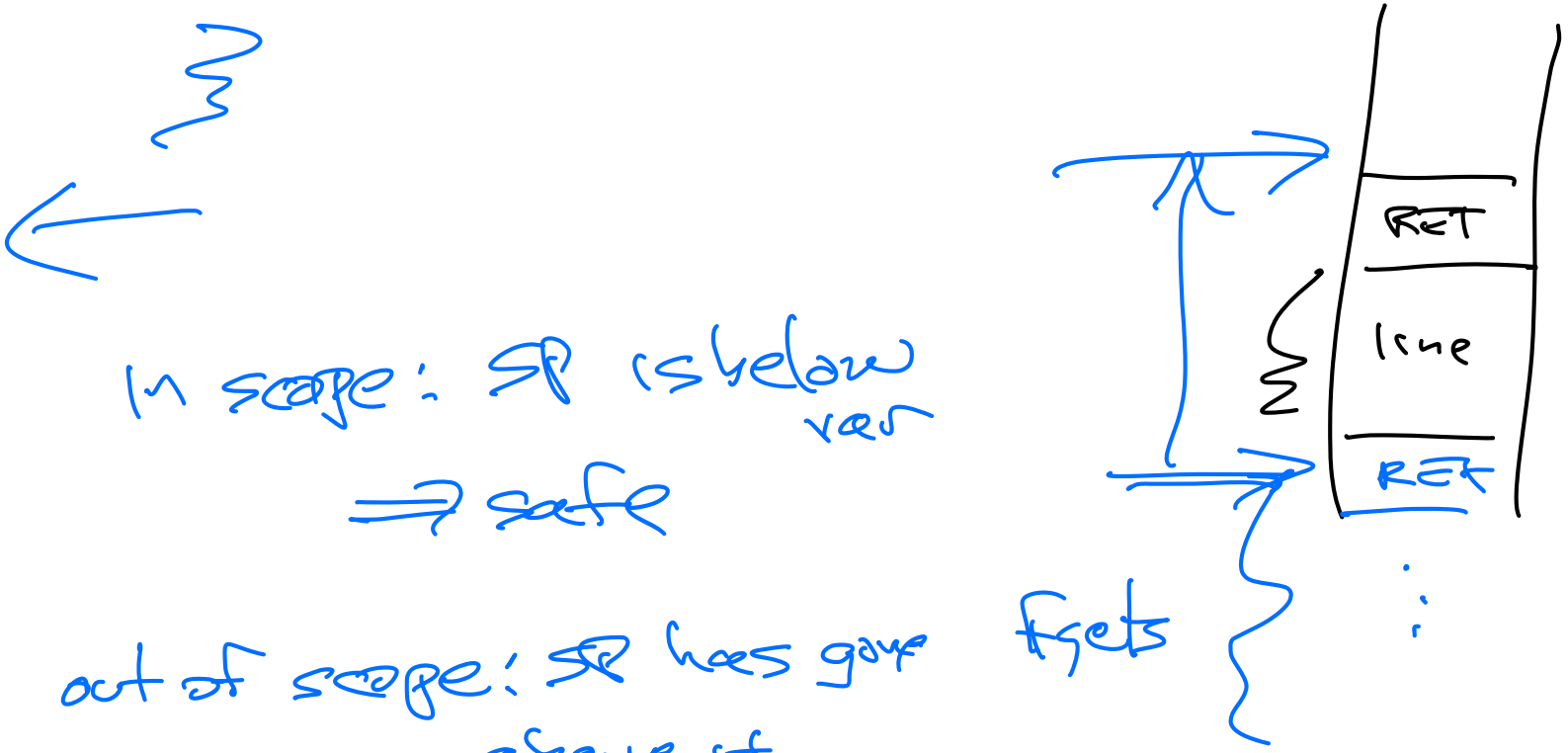
strAddr



{

char line[128]

→ fgets(line, sizeof(line), stdin)



In scope: SP is below var

⇒ safe

out of scope: SP has gone above it

⇒ will get re-used