Notes for 9/4/09

To see memory map of a process on linux, type:
`cat /proc/<PROCESS ID>/maps`

Note that user processes all have their own virtual address space. We will discuss what this means later in the class, but an implication of this is that processes generally cannot see the memory of other processes and they can behave as though they have the address space all to themselves.

On 32-bit MIPS linux, this is how our address space is organized:

Memory addresses ↑	0xffffffff	
		reserved for kernel
	0xc0000000	
	0xbfffffff	
		.stack
		stack (grows ↓)
		. . .
		heap (grows ↑)
		. . .
		.data (grows ↑)
	0x00000000	

What is stored in the stack frame:
Local variables
`$t#` – **temporary registers** / **caller-saved** – you have to save these to the stack before a function call and restore them to the register after a function call if you want them to necessarily have the same value, since the callee can overwrite them
`$s#` – **saved temporary registers** / **callee-saved** – you have to save these registers to the stack before you use them and restore them for the caller before your function returns
`$fp` – **caller frame pointer**
`$gp` – **global pointer**
`$ra` – **return address**

SPIM calling convention (MIPS linux will differ):

Memory addresses ↑		Argument 4 (allocated by caller)
		Argument 3 (allocated by caller)
		Argument 2 (allocated by caller)
		Argument 1 (allocated by caller)
	`$fp` →	Saved registers (allocated by us)
		Local variables (allocated by us)
	`$sp` →	Arguments for functions we call...

How to generally write MIPS assembly for function calling (we will learn MIPS linux specific convention later):

```
f:
    addi $sp, $sp, -24 // reserve space for stack frame
    sw $ra, 20($sp)    // store return address to stack
                       // (if we call other functions, $ra will be
                       // overwritten)
    sw $fp, 16($sp)    // store previous frame pointer to stack
    addi $fp, $sp, 24  // update frame pointer
    sw $a0, 32($fp)    // save argument to stack
                       // we do this into stack space allocated by
                       // caller

    . . .              // do stuff

    lw $ra, 20($sp)    // restore return address
    lw $fp, 16($sp)    // restore frame pointer
    addi $sp, $sp, 24  // move stack pointer back
    li $v0, 0          // set function return value
    jr $ra             // jump to return address
```