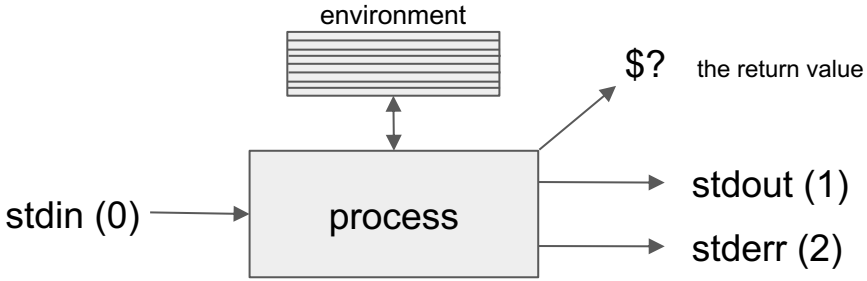
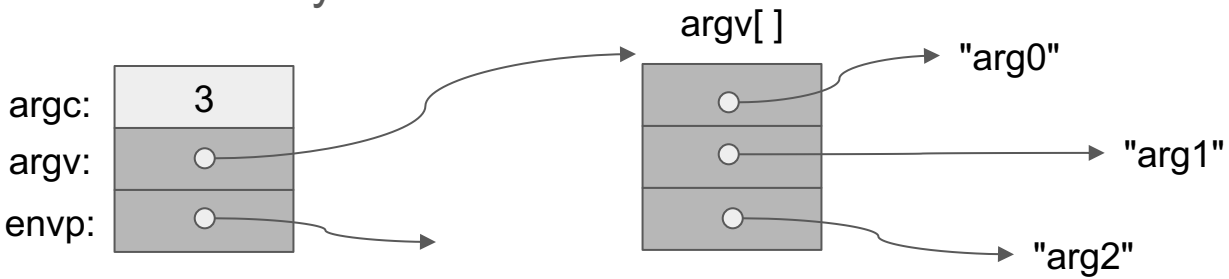
The Process and Standard File Descriptors (fds)

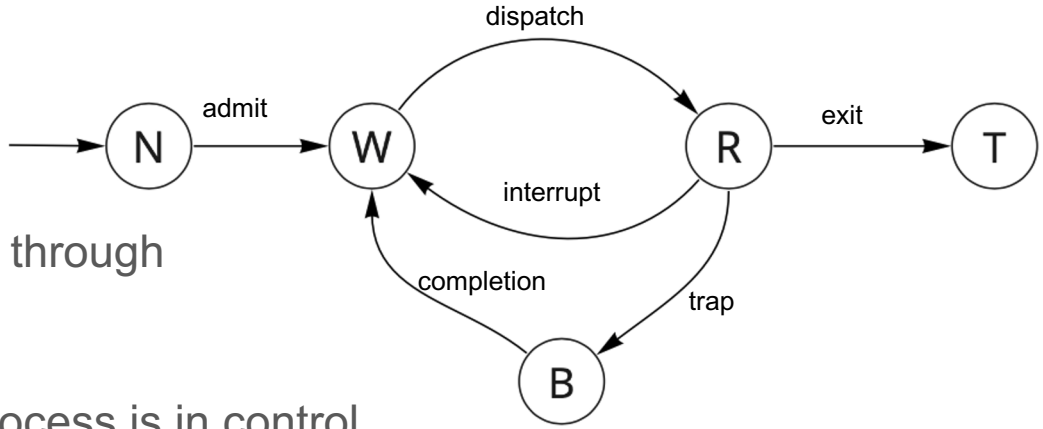
- A process is a running program
- From the CLI:
  - java program arg0 arg1 arg2
  - program arg0 arg1 arg2
- From within the program
  - `public static void main(String[] args)  //Java`
  - `void main(int argc, char* argv[], char** env)// C`
- Variables within memory:



Java Parlance:
System.in == stdin
System.out == stdout
System.err == stderr



Process Status Diagram



- Control of the computer moves through a well-defined cycle

- At any point in time, a single process is in control
  - loosely speaking a process is equivalent to a program

- Transitions:
  - admit: A request is made to allow your program to content for control
  - dispatch: Your program is given control
  - exit: Your program asserts that it is done
  - interrupt: The OS seizes control
  - trap: Your program (implicitly or explicitly) requests a service to be performed
  - completion: The request is satisfied

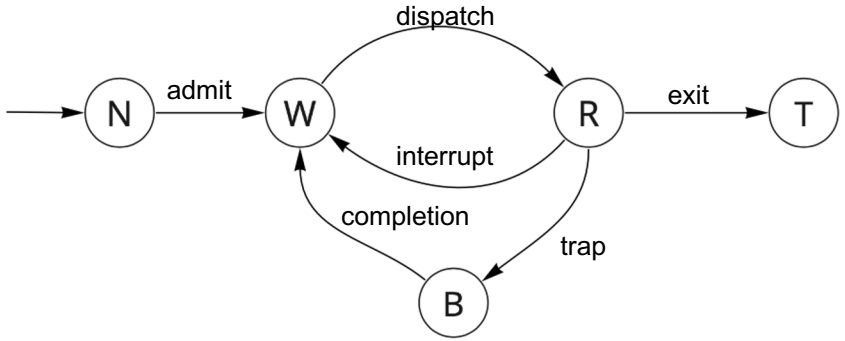
MIPS ISA Architecture: OS interface

```
# Print the integer ‘1’
# Macro: print_di %imm
li $a0, %imm
li $v0, 1
syscall
```

- Service Requests to the operating system via the ‘syscall' instruction

Service Name	$v0	input: $a0..$a3	output: $v0..$v1
print integer	1	$a0 = value	none
read integer	5	none	$v0 = value
malloc	9	$a0 = size	$v0 = buffer address
exit	10	none	none
file read	14	$a0 = fd, $a1 = buffer address $a2 = num bytes	$v0 = bytes read -1 == error 0 == eof

Execution of your program



1. Invoke the program:
2. Wait to use the CPU
3. Execute for as long as you can -- Until
    - (Exit) You are done
    - (Interrupt) You get interrupted by some outside force
    - (Trap) You need help because you made an error or you requested it
4. If you were interrupted, goto Step 2
5. If you trap, and then goto Step 2
    - recover from the error, or
    - obtain the requested server

Analogy: Driving your Car from LA to Vegas

Interrupts and Traps: (results in the kernel seizing control)

- Interrupts are asynchronous events
  - such events occur outside of your process/program
  - such events may or may not be associated with your program
  - Examples:
    - data has arrived on the NIC
    - a disk request for a different process has been completed
- Traps are synchronous events
  - such events occur inside of your process/program
  - some events are error conditions, e.g.,
    - division by zero
    - invalid or illegal memory access
  - some events are requests, e.g.,
    - read/write from a file
    - create a child process
- Exits are a specific type of trap that results in a different flow through the PSD

➢ For speed, traps are to be avoided!

"read" system call

```
int retval;
int fd;
fd = open("/home/steve/filename", O_RDONLY);
```

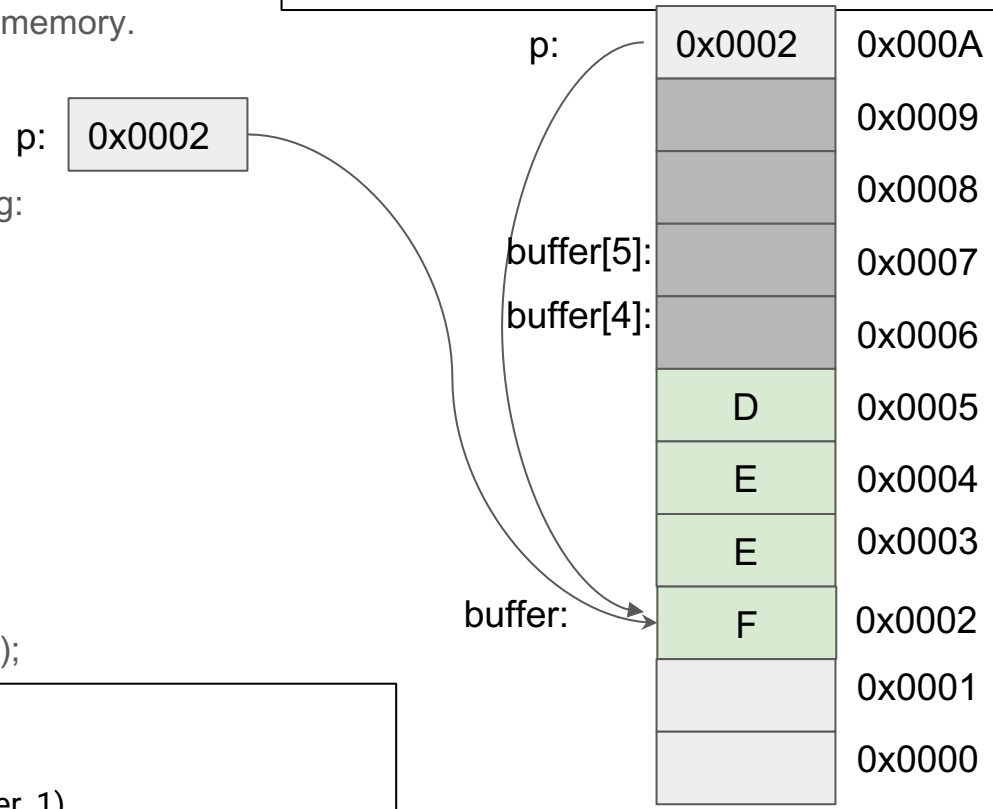
- You need to allocate a buffer, a block of memory.

```
byte buffer[8];
int * p = &buffer;
```

- Make a read request to the OS, providing:
  - the identifier of the file to read
  - the location of the buffer
  - the number of bytes to read

```
retval = read(fd, &buffer, 8);
```

- What are the values passed to read?
- Value of retval informs what happened.
  - retval == -1: error
  - retval == 0: end of file
  - retval <= 8: number of bytes read
- Cast the code
  - retval = read(fd, (void *) &buffer, 8);




```
Java:  File fd = new scanner("/home/steve/filename");
       buffer = fd.nextByte();        //  in C: read(fd, &buffer, 1)
```

Reading a block of 10 bytes!

- Java Example:

```
byte header[10];
stdin = new Scanner(System.in);
for( i = 0; i < 10 ; i++ ) {
    header[i] = stdin.nextByte();
}
```

- The Scanner class only handles primitive types
- We are not in a position to reimplement it.
- The OS knows nothing about my Java class
- Consequently, this results in 10 systems calls

- Equivalent C Example

```
byte header[10];
for( i = 0; i < 10 ; i++ ) {
    header[i] = (byte) getchar();
}
```

```
retval = read(fd, &header, 10);
```

A More Efficient Approach

- Read does not care what it is reading
- This results in 1 system call
- But we need to understand pointers: * and &
- Moreover we need to cast our variables

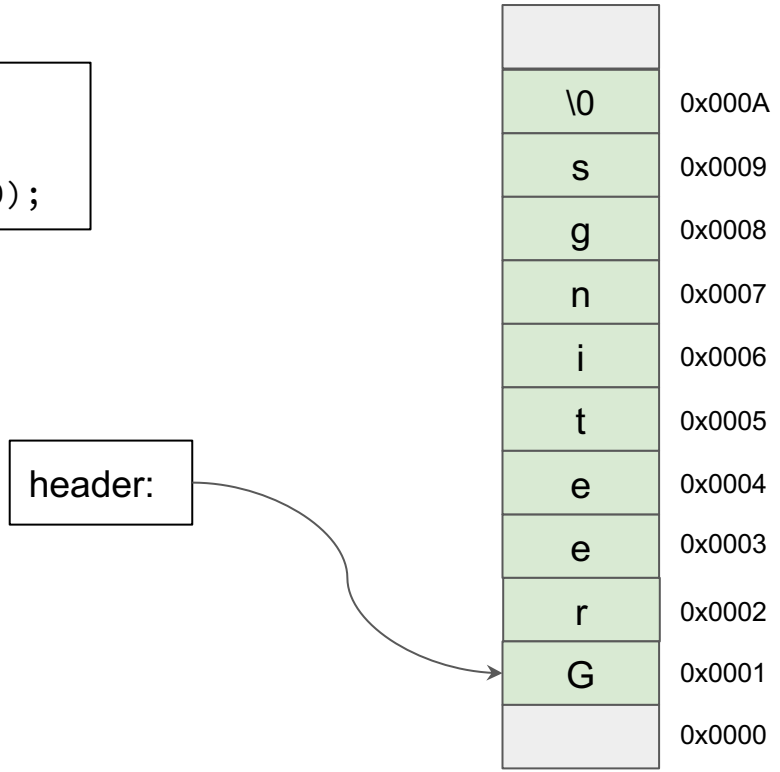
- Via 'read' system call (1 trap):

```
byte header[10];

retval = read(STDIN_FILENO, (void *) &header, 10);
```

- Java Example (10 traps):

```
byte header[10];
stdin = new Scanner(System.in);
for( i = 0; i < 10 ; i++ ) {
    header[i] = stdin.nextByte();
}
```

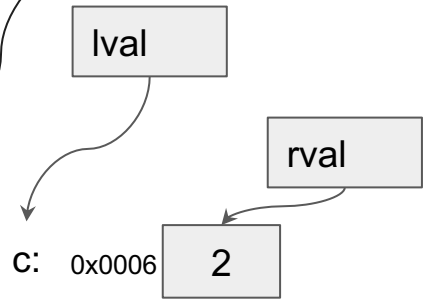


Variables: Names and Values

1. determine the lval for “c”
2. extract the rval of “c” at MEM[lval]
3. add 1 to that value
4. Write the value from #3 into MEM[lval]

- We all know what variables are, right!
- A variable has two values: a lval and a rval
- Consider the following assignment:

c = c + 1



	0x000A	0
	0x0009	0
d:	0x0008	5
	0x0007	0
c:	0x0006	2
	0x0005	
b:	0x0004	6
p:	0x0003	?
	0x0002	0
a:	0x0001	1
	0x0000	42

- The lval of c is the address in memory
- The rval of c is the value located at that address in memory

- The rval can be a "integer", "float", "char", etc.
- If the rval is an address, than it is known as a "pointer"

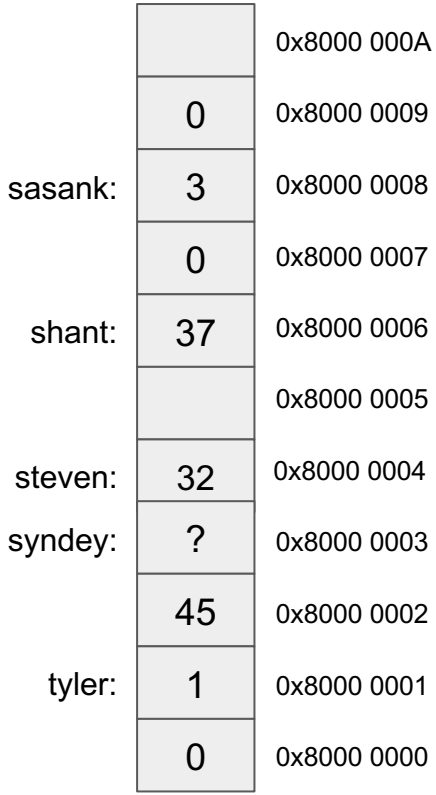
int * p = & c;

p: ?

- Hold that thought! (* p) == [ ]

Memory

- We all know what an array is right!
  - Memory is just an array of integers (from 0..255):
    - mem[index] = value
- Do you know what an associative array is?
  - It's just an array that stores both the lval and rval of a variable:
    - array[*"name"*] = value; mem["steven"] = 32
  - You use "name" to lookup the appropriate index
- Consider the memory to the right

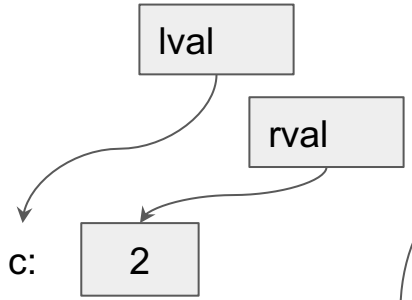


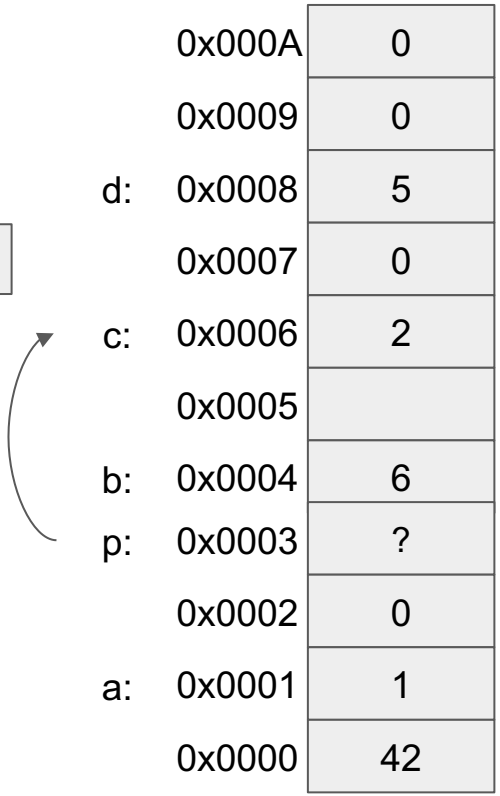
* * * p == 42

Variables: Names and Values

- We all know what variables are, right!
- A variable has two values: a lval and a rval
- Consider the following assignment:

```
c = c + 1;
```





- The lval of c is the address in memory
- The rval of c is the value located at that address in memory

- The rval can be a "integer", "float", "char", etc.
- If the rval is an address, than it is known as a "pointer"

```
int * p  = & c;
```

- Hold that thought! (* p) == 2

- What is the address c? What is the value of p? What is the address of p?