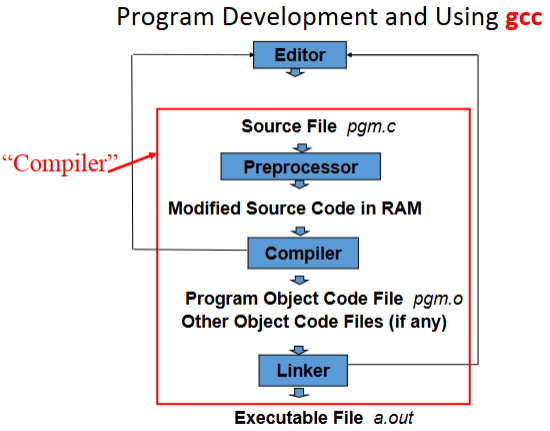
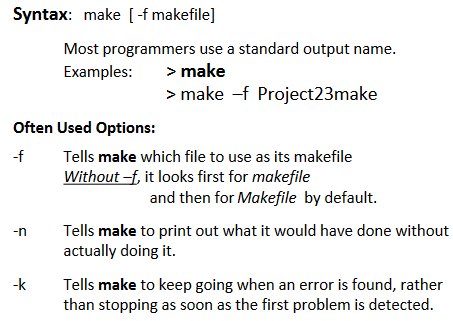
**Study: 4-X slide 23**



**Makefiles** –know the terminology and purpose (5-X)

* In a large project, one doesn’t want to re-compile everything, every time a change is made.
* **Make** keeps track of dependencies,
  + Of what needs re-compiling,
  + Of what needs re-linking
* The touch command:
  + Syntax: touch [options] *files*
  + Changes two timestamps associated with a file:
    - Its modification time (when the file’s data was last changed)
    - Its access time (when the file was last read)



**Linux commands:**

* **chmod** –change the access permissions to files system objects like files and directories.
  + Syntax: chmod *options permissions filename*
* ***ps*** –
* **pwd** –print name of current/working directory.
* **wc** –

**How to get help on a Linux command?**

* **help** command – Gives you more information about any command or all if you leave out command

File permissions: What are the 3 categories of users and three categories of permissions?

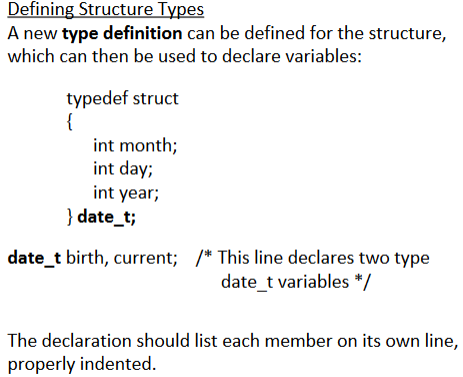
* Users
  + User/owner **u**
  + Group **g**
  + Other (the world, everybody else) **a**
* Permissions
  + Read **r** or **4**
  + Write **w** or **2**
  + Execute **x** or **1**

**Know the name of the debugger and its commands**

* **gdb** is the GNU Project debugger
* **-g :** compile with the **–g** flag to set up for debugging
* **break** place: place can be the name of a function or a line number
  + Ex: break main will stop execution at the first instruction of tour program
* **run** command-line-arguments
  + Begin execution of your program with arguments
* **c**
  + continue running your program (after stopping, e.g. at a break-point).
* **delete** N
  + Removes breakpoints, where N is the number of the breakpoint
* **step** 
  + Executes current instruction and stops on the next one
* **next**
  + Same as **step** except this doesn’t step into functions
* **print** E
  + Prints the value of any variable in your program when you are at a breakpoint, where E is the name of the variable you want to print
* **quit**
  + When time to exit gdb

**Know how to use *printf()* which writes. The decimal point is a character and takes up space!**

**Write a *typedef struct*; and declare a variable using that type**

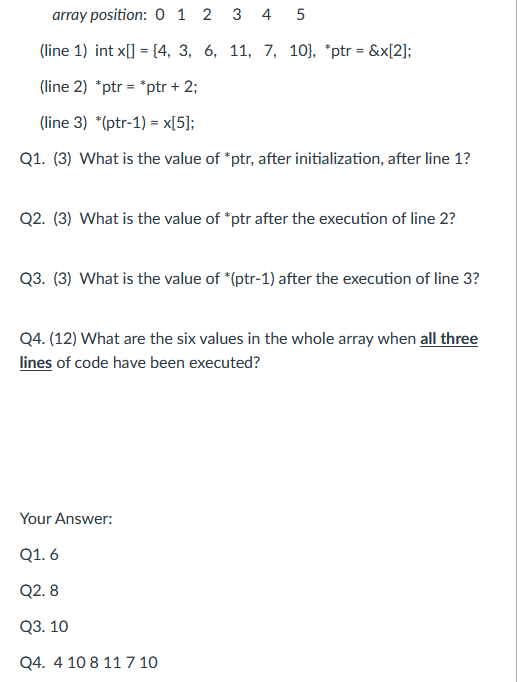


* Common Industry Practice:
* The “\_t” suffix is not require by C, but it certainly makes it easier to keep a program readable

**Review string functions:**

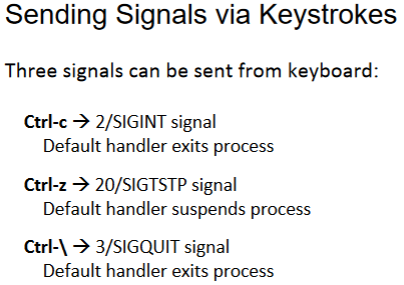
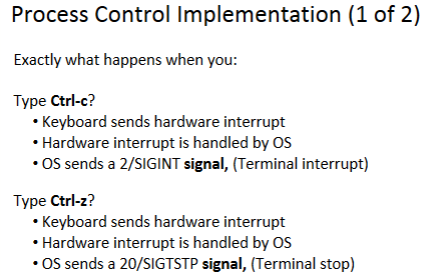
* **strcat**(s, t); – concatenates string t to the end of s
* **strcpy**(s,t); – copies string t onto and over string s

**Questions on pointers. You will have an array and have to figure out what is in the array after 3 lines of code, like first exam but with different code and different data.**

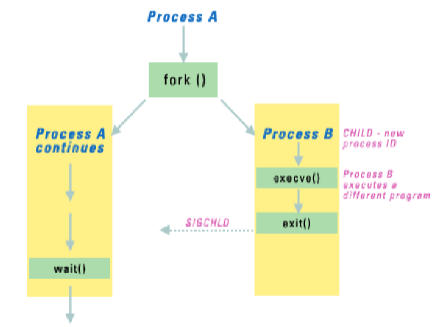


**System Calls:**

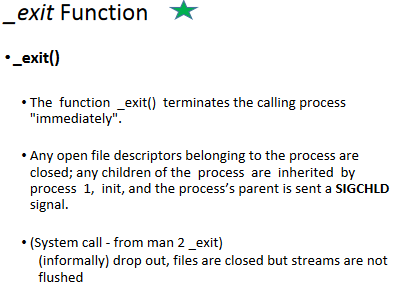
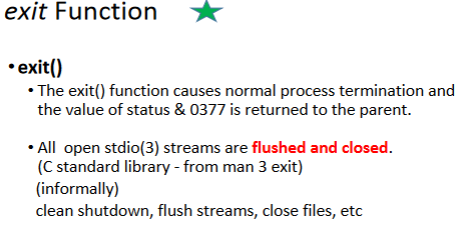
* Know how to send interrupt-signals to a program and what they each do:

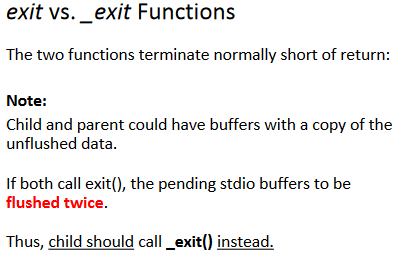


* Know the names, know the definitions: and the use, do not need to know flags and permissions. What kind of interrupt they send to the kernel, hardware or software interrupt? (7-X)
* A **system cal**l is an explicit request to the kernel made via a **software interrupt**.
  + **open()** 
    - opening a file informs the kernel that an application wants to access a file
    - Allows the kernel to set aside resources
    - Returns file descriptor on success, or –1 on error
  + **close()** 
    - Closing a file tells the kernel it may free resources associated with managing the file
    - Close returns 0 if OK, -1 if error
  + **read()** 
    - Copies at most **count** bytes from the current file position to **buffer** and updates the file position
    - Returns the number of bytes read
      * Returns <0 if error
      * Returns 0 if end-of-file (EFO) occurs
    - Read may return fewer bytes than requested (short reads)
  + **write()** 
    - Copies at most **count** bytes from **buffer** to the file position and updates position
    - Returns the number of bytes written
      * Returns <0 if error
    - It is possible that fewer bytes were written than requested (short writes) this is not an error, but certainly a challenge to deal with
  + **lseek()** 
    - Causes the logical position in the file to change
      * i.e. where the next read or write will commence from
      * Also referred to as Changing the File Offset
    - Whence determines how position will change:
      * SEEK\_SET : pointer is set to offset bytes.
      * *SEEK\_CUR :* pointer is set to its current location plus offset.
      * SEEK\_END : pointer is set to the size of the file plus offset.
* Know the flow of process between parent and child after a fork. Child cannot have more than one parent, parent can have multiple children
  + **fork()** creates a new process, the child, which is an almost exact duplicate of the calling process, the parent.
    - The child has its own process ID.
    - The child inherits the same stack, data and heap.
    - After fork(), both processes can make changes independently.
    - In parent: return process ID of child on success, or –1 on error;
    - If successfully created, child always returns a 0.



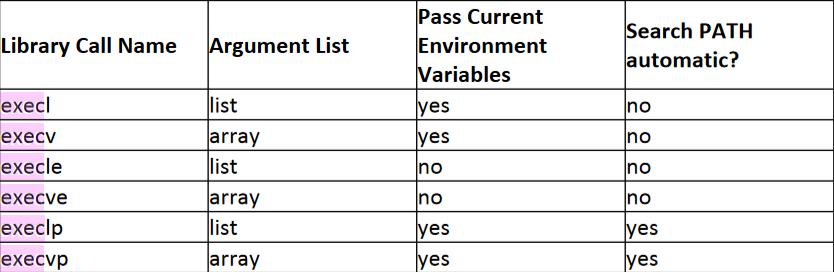
* Know the difference between exit() and \_exit() (12-X and lab10 slides)

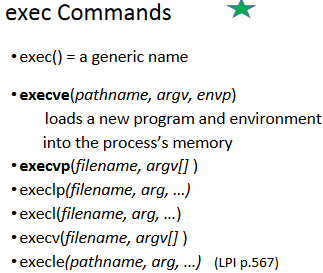


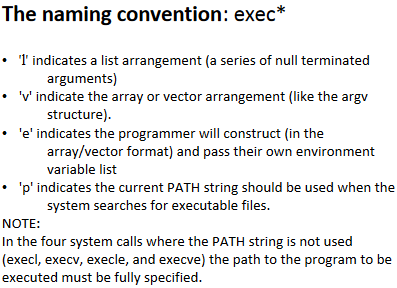


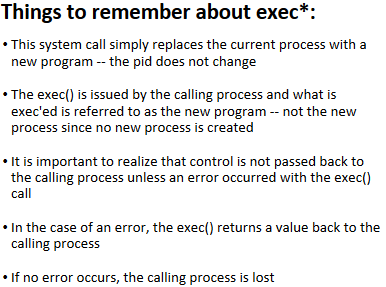
**Words & commands to know:**

* exec() (9-X)

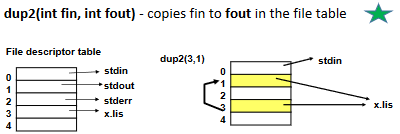




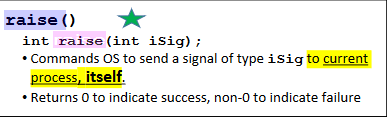




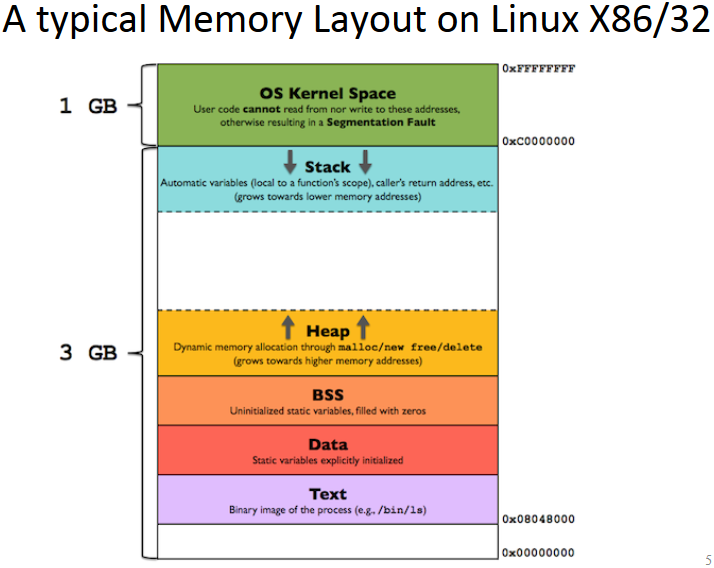
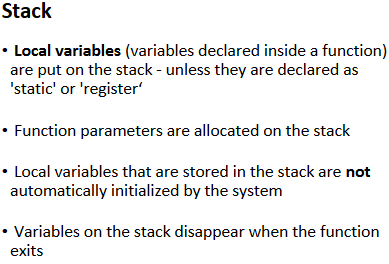
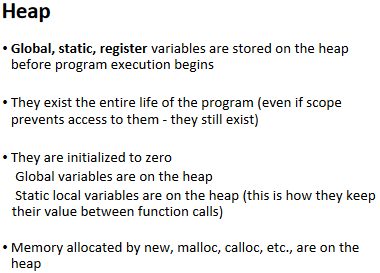
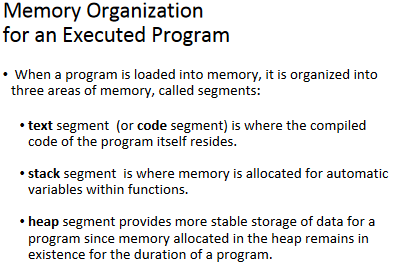
* file descriptor table



* I-node table
* open-file table
* fork()
* init (9-X fork exec)
* IPC
* UID
* RUID
* SUID
* EUID
* PID
* man
* pipe & their file descriptors
* ps
* shell
* POISX
* a zombie process
* wrapper routine & sys\_handler (6-X)
* wait and waitpid
* The system call: **raise()**



* Top level understanding of pipes, signals, shared memory, messages -> all are IPC
* Where does a file descriptors come from, who sets it, who puts a value in it. EX: fd=open()
* Know which IPC system calls have automatic synchronizations and which one does not (12-X)
  + Inter-Process Communication (IPC) Pipe
* Reviews the System I/O calls, all five of them covered in class.
* Default File Descriptors:
  + 0: standard in
  + 1: standard out
  + 2: standard error
* What is the organization of memory in Linux...heap...stack...etc. (13-X)



* Familiar with names of shared memory calls, and be able to pick from 4 choices which one is not a memory call
* Know about argc and argv[], how they work (8-X)
  + **argc** holds the total number of command line arguments (implicit+explicit).
  + **argv** is an array of pointers to strings. Each argument value is stored as a string.