Items highlighted in purple are on the exam

Review syntax #define #include

- Writing #define (constants, C-1 slide 51-52)
  - #define PI 3.14159
  - #define MONTHS_IN_YEAR 12
- #include <stdlib.h>

Printing doubles

- printf("Height is %6.2f\nLength is %6.2f\n", height, length);
  - 6 refers to width total
  - 2 refers to precision
- *What will be printed?*
  - Double k= 6.789
  - printf("%4.2f", k)
  - >> 6.79

Makefiles - know terminology (5Unix, slide 3)

Make

- Keeps track of what needs to be recompiled and relinked

Make commands

- -f
  - Tells **make** which file to use as its makefile, without -f it looks for the first makefile
- -n
  - Tells **make** to print out what it would have done without actually doing it
- -k
  - Tells **make** to keep going when an error is found, rather than stopping as soon as the first problem is detected

---

Know how to use gcc with and without –o

- gcc *FileName.c*
- gcc *FileName.c* -o *FileName*

System calls, know the definition and the use, don't need to know flags and permissions (9-unix) for i/o

open(): open new or existing file for reading and writing, truncating to zero bytes; file permissions read+write for owner, nothing for all others (slide 7)

  - Returns file descriptor (fd)
  - e.g. int open (const char **pathname*, int *flags* … /*mode_t *mode* */);

close(): closing a file tells the kernel it may free resources associated with managing the file (slide 11 for example)

  - Close returns 0 if ok, -1 if error

read (): returns number of bytes read, 0 on EOF (end of file), or -1 on error (slide 13 for example)

  - Each open file has a notion of a current position in the stream of bytes
  - read() copies at most count bytes from the current file position to buffer and updates the file position
  - 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
  - Possible that fewer bytes were written than requested (short writes)

lseek (): returns new file offset if successful, or -1 on error

- Causes logical position in the file to change
    - i.e. where the next read or write will commence from
    - 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

what system call is used to redirect data flow from one file descriptor or another

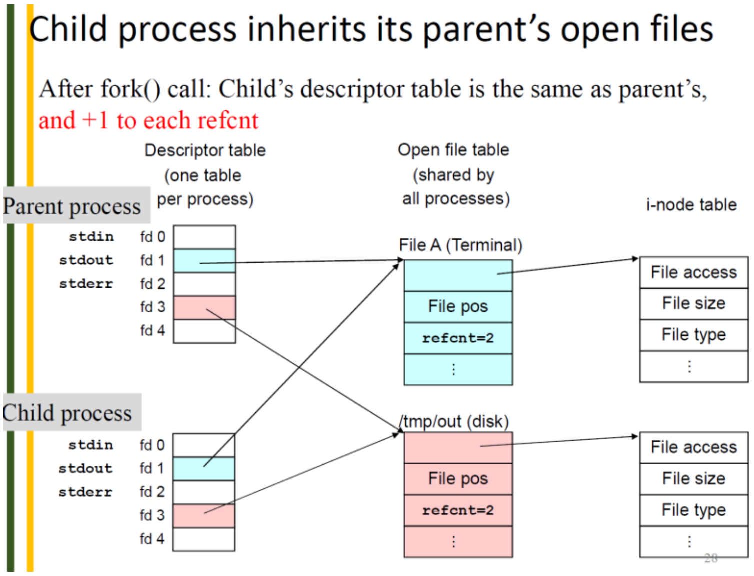
-

what system call allows one process to create a new process

- fork()

Know a flow of process between parent and child after a fork

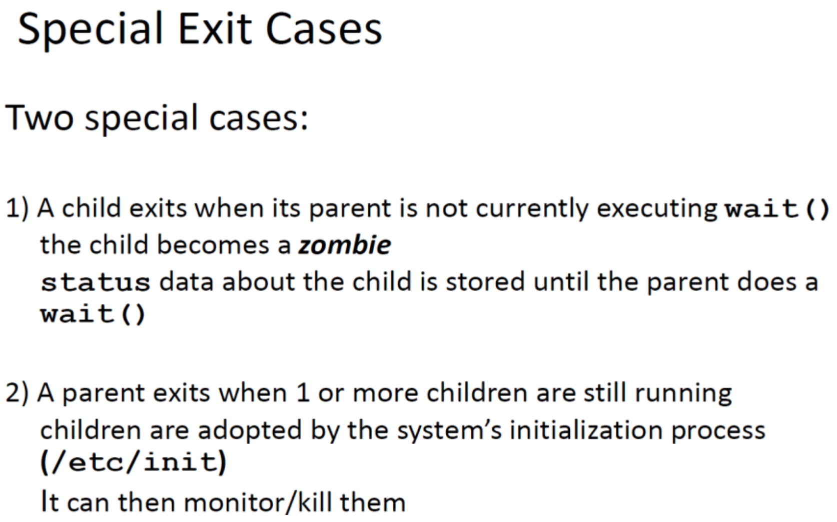
- Parent-Child relationship with the file.
    - The parent opens the file with the fd.
    - The child inherits the open file from the Parent.
    - The child can close the file, but it will still be open by the parent.



Child process inherits its parent's open files

After fork() call: Child's descriptor table is the same as parent's, and +1 to each refcnt

How many child processes can a parent have? How many parents can a child have?



Special Exit Cases

Two special cases:

1) A child exits when its parent is not currently executing `wait()`
the child becomes a ***zombie***
`status` data about the child is stored until the parent does a `wait()`

2) A parent exits when 1 or more children are still running
children are adopted by the system's initialization process (`/etc/init`)
It can then monitor/kill them

Study 4-unix, slide 16

1. Editor
   - Source File: *pmg.c*
2. Preprocessor (4-Unix, slide 17)
   - Modified Source Code in RAM
3. Compiler (4-Unix, slide 18)
   - Program Object Code File *pgm.o*
   - Other Object Code files (if any)
4. Linker (4-Unix, slide 19)
   - Executable File: *a.out*

Study 7-unix, slide 3

Command line arguments - format

- exec_filename arg1 arg2 arg3
  - Arguments are listed after executable name
  - Arguments are separated with whitespace

Study 13-unix, slides 2-5

Areas of Memory, Segments (slide 2):

- Text segment (code segment): where compiled code of the program resides
- Stack segment: where memory is allocated for automatic variable within functions
- Heap segment: provides more stable storage of data for a program since memory allocated in the heap remains in existence for the duration of a program

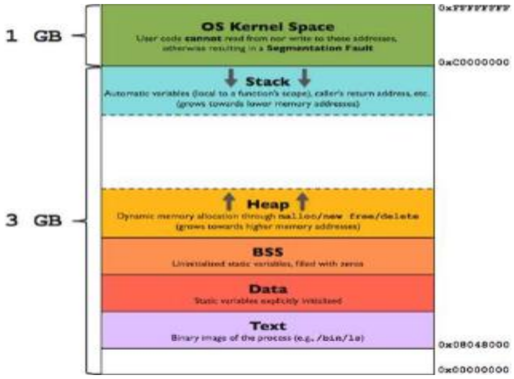
Stack (slide 3)

- Local variables are put on the stack - unless they are declared as ‘static’ or ‘register’
- Function parameters are allocated on the stack
- Local variables that are stored in the stack are not automatically initialized by the system
- Variables on the stack disappear when the function exits

Heap (slide 4)

- Global, static, register variables are stored on the heap before program execution begins
- They exist the entire life of the program (even if scope prevents access to them - they still exist)
- Initialized at zero
  - Global vars are on the heap
  - Static local vars are on the heap (this is how they keep their value between function calls)
- Memory allocated by new, malloc, calloc, ect., are on the heap

Slide 5 – A Typical Memory Layout on Linux X86/32


0xFFFFFFFF
1 GB
OS Kernel Space
User code cannot read from nor write to these addresses, otherwise resulting in a Segmentation Fault
0xC0000000
Stack
Automatic variables (local to a function's scope), caller's return address, etc. (grows towards lower memory addresses)
3 GB
Heap
Dynamic memory allocation through malloc/new free/delete (grows towards higher memory addresses)
BSS
Uninitialized static variables, filled with zeros
Data
Static variables explicitly initialized
Text
Binary image of the process (e.g., /bin/ls)
0x08048000
0x00000000

OS Kernel Space (User can not read or write to)
↓Stack↓ (Automaic varables local to func scope)
↑Heap ↑ (Dynamic memory allocation)
BBS (Uninitalized static variables, filled with zeros)
Data (Static Variables explicity initialized)
Text [Binary image of the process(/bin/ls)]

Standard File descriptors (9-unix, slide 4)

- 0: standard in.
- 1: standard out.
- 2: standard error

Top level understanding of pipes, signals, shared memory -> all are IPC

Know what IPC stands for (12-unix pipes)

  - Inter-process communication

Know the difference between exit() and _exit() (In the Pipe slides (12-unix, slide 29-31) and the slides for lab10)

exit()

- Causes normal process termination and the value of the status & 0377 is returned to the parent
- All open stdio(3) streams are flushed and closed (C standard library - from man 3 exit)
- Clean shutdown, flush streams, close files, ect.
- E.g. void exit(int status);

_exit()

- Terminates the calling process "immediately"
- Any open file descriptors belonging to the process are closed; any children of the process are inherited by process 1, init, and the process's parent is sent a SIGCHLD signal
- System call - from man 2_exit
  - (informally) drop out, files are closed but streams are not flushed

Exit vs _exit

- Two functions terminate normally short of return:
  - Child and parent could have buffers with a copy of the unflushed data
  - If both call exit(), the pending stdio buffres to be flushed twice
    - The child should call _exit() instead

Know which system calls have automatic synchronizations and which don't

Know how to use printf()

- Int k = 97, a = 5, b = 9
- printf("value of k: %i", k); %d can be used in place of %i
- printf("%i%i\n\n", a, b); (More ex's on C-1, slides 57-62)

Go over printf() slides: right adjusted, left adjusted, hex, octal, and so forth

List of conversion specifiers: (C-1, slide 45)

○ Octal -	%o
○ Hexadecimal -	%x
○ Left adjusted -	%-
○ Right adjusted -	%+
○ Zero filled	%0

Know the syntax of getChar() and putChar(); (C2)

- **Putchar** - print a space (C-2, slide 7-8)
  - putchar(32);
  - putchar(' ');
  - #define SPACE ' '
    putchar (SPACE);

- **Getchar** - gets char from inputs (C-2, slide 9)
  - c1 = getchar();
  - c2 = getchar();
  - putchar(c1);
  - putchar(c2);
  - putchar(‘\n’)
  - a b (NL) from keyboard
  - a b (NL) from putchar

Environment variables: HOME, PATH, etc.

Know the abbreviations of: IPC, UID, PID,

- IPC - Inter-process communication
- UID - user identifier
- PID - process identification number

Know the command to change file access at the keyboard

- chmod

Know the linux call that prints path name:

- pwd

Know how to rename a file

- mv file1 file2

How to remove empty directories :

- rmdir

Go over how we create Linux file names: which characters you can and cannot use

- **/ ← ILLEGAL IN LINUX!!!!**
- **A (CAPITAL LETTERS)**
- **. (DASHES)**
- **_ (UNDERSCORES)**

Know the name of the debugger and its commands (4-Unix, slides 22-25)

- gdb - GNU Project debugger
  - Compile with -g flag to set up for debugging
- gdb commands (4-Unix, slide 24)
  - break *place*
    - Place can be the name of a function or a line number

File permissions: What are the 3 categories of users (guru99.com)

- User: Owner of the file
- Group: All users belonging to a group will have the same access permissions to the file
- Other: any other user who has access to a file

Where does a file descriptor does come from, who sets it, who puts a value in it. EX fd=open()

- Parent closes file descriptors, child executes it

Familiar with names of shared memory calls, and be able to pick from 4 which isn’t a memory call

- Create/Access Shared memory
  - id = shmget (KEY, Size, IPC_CREAT | PREM)
- Deleting Shared Memory

    - i = shmctl (id, IPC_RMID, 0)
    - Or use ipcrm
- Accessing Shared Memory
    - memaddr = shmat(id, 0, 0)
    - memaddr = shmat(id, addr, 0)
    - memaddr = shmat(id, 0, SHM_READONLY)
        - System will decide address to place the memory at
    - Shmdt (memaddr)
        - Detach from share memory
- Message queue (13-unix, slide 36 start)

Know how to send interrupt signals to program: ctrl^c ctrl^z ....(11-unix, slide 10)

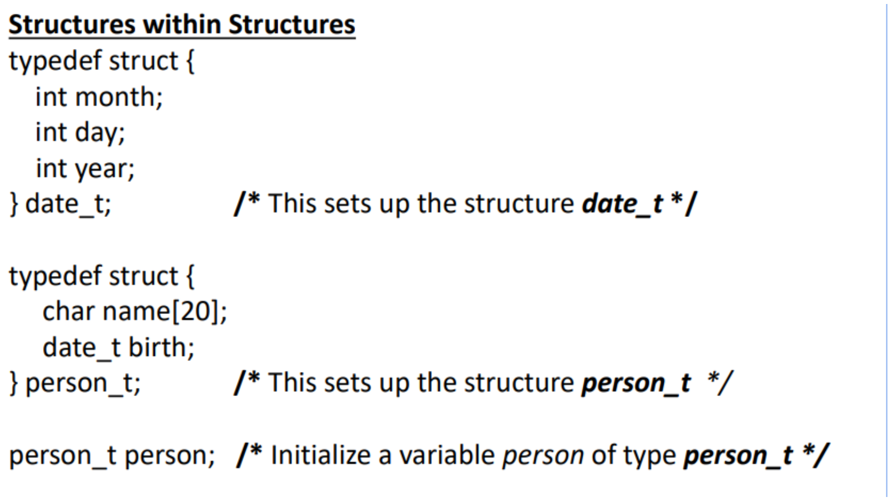
- Type Ctrl-c
    - Keyboard sends hardware interrupt
    - Hardware interrupt is handled by os
    - OS sends a 2/SIGINT signal
    - Default handler exits process
- Type Ctrl-z
    - Keyboard sends hardware interrupt
    - Hardware interrupt is handled by OS
    - OS sends a 20/SIGTSTP signal
    - Default handler suspends process
- Type Ctrl-\
    - Default handler exits process
    - Sends 3/SIGQUIT signal
- Sending signals via commands(slide 17)
    - Kill -*signal pid*
        - Send a signal of type signal to the process with id pid
            - Can specify either signal type name (-SIGINT) or number (-2)
- Sending signals via function call
    - raise()
        - Int raise(int iSig):
        - Commands OS to send singal of type iSig to current process, itself
        - Returns 0 to indicate success, non-0 to indicate failure
    - kill()
        - int kill(pid_t iPid, int iSig);
        - Sends an iSig signal to the process whose id is iPid
        - Equivlent to raise(iSig) when iPid is the id of the current process
- Signal types (11-unix, slide 9)

Invalid or valid names of variables in c (C-1, slide 15)

Valid Examples: density
sum3
x_y
x2_2
Volume

Invalid Examples: 2sum
x&y
x-y
X2.2
1Volume

Write a #define, a struct, declare a variable using a struct



Structures within Structures

```
typedef struct {
   int month;
   int day;
   int year;
} date_t;            /* This sets up the structure date_t */

typedef struct {
    char name[20];
    date_t birth;
} person_t;          /* This sets up the structure person_t  */

person_t person;   /* Initialize a variable person of type person_t */
```

Review string functions: strcat, strcpy

strcat: appends a copy of the string pointed by s2 to the end of the string pointed by s1, returns a pointer to s1 where the resulting concatenated string resides

- ○ char *starcat(char *s1, const char *s2);
    - ■ s1: pointer to a string that will be modified
    - ■ s2: pointer to a string that will be copied to the end of s1
- ○ Returns a pointer to s1

strcpy: copies the string pointed to by s2 into the object pointed by s1, returns a pointer to the destination

- ○ char *strcpy(char *s1, const char *s2);
    - ■ s1: an array where s2 will be copied to
    - ■ s2: the string to be copied
- ○ returns s1

Know the meaning of argc and argv

- ○ Argc - the number of parameters passed to your program when it's invoked from command line
- ○ Argv - the array of received parameters, and is an array of strings

Pointers: Is a construct that gives you more control of the computer's memory. It is the memory address of a variable.

```
int x[] = {5,6,4,-8,3,7};  *ptr = 5              x[3]-*ptr = -13
int *ptr = &x[0];          *ptr+3 = 8            *ptr + x[5] + *(ptr + 1) = 18
                           *(ptr+3) = -8         *x = 5
                           *ptr+*(ptr+5) = 12    *x + *ptr = 10
                           *(ptr+2)-1 = 3        x[2]-*ptr+3 = 2
```

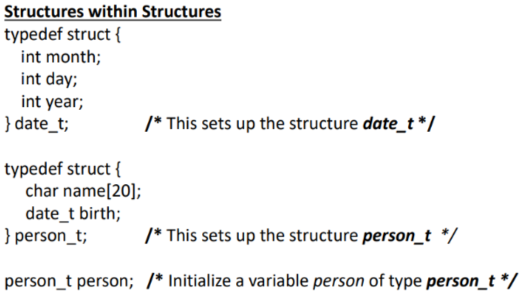
Know the order in which linking, program development, compiling, ect. Happens (4-Unix, slides 16-20)

1. Editor
   - Source File: *pmg.c*
2. Preprocessor (4-Unix, slide 17)
   - Modified Source Code in RAM
3. Compiler (4-Unix, slide 18)
   - Program Object Code File *pgm.o*
   - Other Object Code files (if any)
4. Linker (4-Unix, slide 19)
   - Executable File: *a.out*

- System calls for i/o
- What system call is a notif. to a process that an event has occurred
- GNU name -> gdb
- Commands for gnu
- A file in Linux can be?

Declare typedef struct named triangle_t

- Int array of side with an array length of NSIDES
- Double named area
- Int named triangle_kind



Structures within Structures

```
typedef struct {
   int month;
   int day;
   int year;
} date_t;          /* This sets up the structure date_t */

typedef struct {
    char name[20];
    date_t birth;
} person_t;        /* This sets up the structure person_t  */

person_t person;  /* Initialize a variable person of type person_t */
```

- Declare a variable of that type, using above struct (no intizalization required)

wc < f1 > f2, what should your code have printed? (assume all files are located correctly, there may be extra answer lines which may be left blank

Argc =

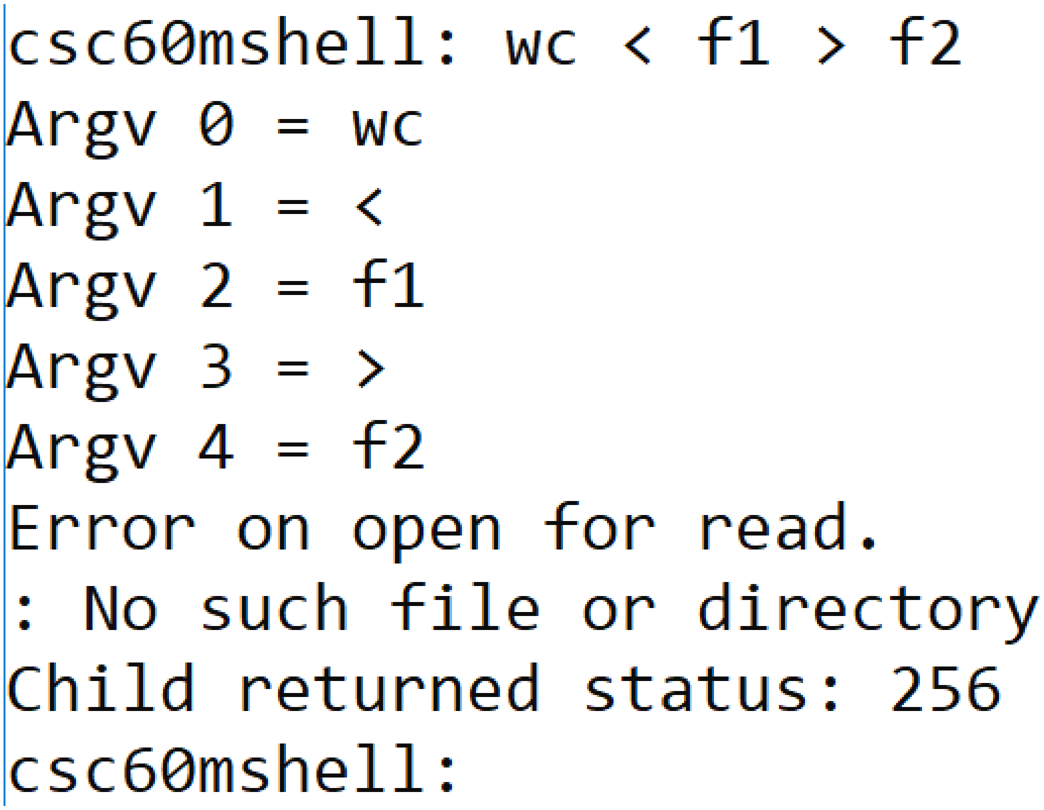
Agrv[1] =

Agrv[2] =

Agrv[3] =

Agrv[4] =

Agrv[5] =



```
csc60mshell: wc < f1 > f2
Argv 0 = wc
Argv 1 = <
Argv 2 = f1
Argv 3 = >
Argv 4 = f2
Error on open for read.
: No such file or directory
Child returned status: 256
csc60mshell:
```