CS241 Final Review Packet

(Answers are provided by students and are NOT guaranteed to be correct.)
Feel free to edit this file

1. C

1. What are the differences between a library call and a system call? Include an example of each.
    a. A system call specifically provides an interface for a process to interact with the kernel service while a library call is simply some function provided by a linked library which may or may not interact with the operating system under the hood.

2. What is the * operator in C? What is the & operator? Give an example of each.
    a. * is the dereference operator; & is the get-address-of operator. Example:
    ```
    Int a = 10;
    Int *b = &a; (&operator; the * in this line represent the pointer type, rather than the * operator)
    *b = 20; (*operator)i
    ```

3. When is strlen(s) != 1+strlen(s+1) ?
    a. strlen(“”) = 0
    1 + strlen(“” + 1) >= 1

4. How are C strings represented in memory? What is the wrong with malloc(strlen(s)) when copying strings?
    a. Character array with a terminating null byte.
    malloc(strlen(s)) does not count for the null byte

5. Implement a truncation function void trunc(char*s,size_t max) to ensure strings are not too long.' Shorter strings are left unchanged. If s is NULL return NULL. e.g. char s[]="abcdefgh; trunc(s,3); puts(s) will print "abc\n".
    a. Void trunc(char *s, size_t max) {
        if(strlen(s) > max)

```
                s[max] = '\0';
        }
B. maybe?
        if (s) {
                S[max] = 0;
        }
```

6. Complete the following function to create a deep-copy on the heap of the argv array. Set the result pointer to point to your array. The only library calls you may use are malloc and memcpy. You may not use strdup.void duplicate(char**argv, char***result);
    a.
```
void duplicate(char **argv, char ***result) {
  int i, j;
  char **curr = argv;

  for(i = 0; *curr != NULL; ++i)
    ++curr;

  *result = malloc(sizeof(char*) * (i+1));

  for(j = 0; j < i; ++j) {
    (*result)[j] = malloc(strlen(argv[j]) + 1);
    memcpy((*result)[j], argv[j], strlen(argv[j]) + 1);
  }
  (*result)[j] = NULL;
}
```
7. Write a program that reads a series of lines from stdin and prints them to stdout using fgets or getline. Your program should stop if a read error or end of file occurs. The last text line may not have a newline char.
    a.
```
int main() {
  char *line = NULL;
  size_t len = 0;
  ssize_t read;

  while((read = getline(&line, &len, stdin)) != -1) {
    printf("%s", line);

  free(line);
```

```
    return 0;
}
```

2. Memory

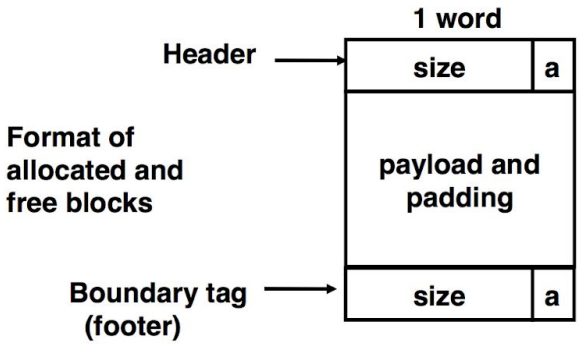
8. Explain how a virtual address is converted into a physical address using a multi-level page table. You may use a concrete example e.g. a 64bit machine with 4KB pages.
    a. The lower 12 bits are converted to the offset bits. The upper bits are sliced into * parts if there are * levels (There might be unused bits). The most upper part refers to the directory index. The second part refers to the hsub-table index.

Nice reference: http://www.cs.cmu.edu/~213/lectures/18-vm-systems.pdf

9. Explain Knuth's and the Buddy allocation scheme. Discuss internal & external Fragmentation.
    a. Knuth’s and Buddy allocation scheme:

Implicit list: Bidirectional coalescing

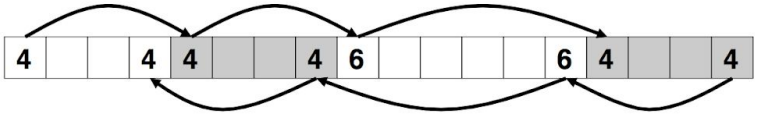
- *Boundary tags* [Knuth73]
  - Replicate size/allocated word at bottom of free blocks
  - Allows traversing a "list" backwards, but requires extra space
  - Important and general technique!

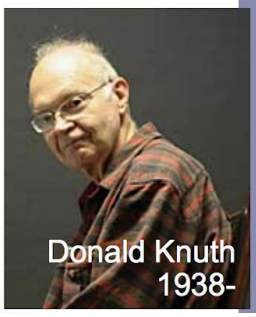


a = 1: allocated block
a = 0: free block

size: total block size

payload: application data (allocated blocks only)





Knuth's boundary tag makes constant time bidirectional coalescing possible.

Buddy allocation: ; ‘

https://en.wikipedia.org/wiki/Buddy_memory_allocation

b. Internal fragmentation is the wasted space within each allocated block because of rounding up from the actual requested allocation to the allocation granularity.

External fragmentation is the various free
spaced holes that are generated in either your memory or disk space.

External fragmented blocks are available for allocation, but may be too small to be of any use.

10. What is the difference between the MMU and TLB? What is the purpose of each?
    a.

    Memory Management Unit(MMU)
    – Hardware unit that translates a virtual address to a physical address
    – Each memory reference is passed through the MMU
    – Translate a virtual address to a physical address

    Translation Lookaside Buffer(TLB)
    – Essentially a cache for the MMU's virtual-to-physical translations table
    – Not needed for correctness but source of significant performance gain

11. Assuming 4KB page tables what is the page number and offset for virtual address 0x12345678 ?
    a. 0x678 is the offset, 0x12345 is the page number (Assume single level page table)

    Add: 32bits
    4*1024 = 2^12
    12/4 = 3

    CAN SOMEONE EXPLAIN ^?

Notice that the address is in hex value. Each entry of a hex value taken 4 bit. 3*4 = 12. Hope this would help.

12. What is a page fault? When is it an error? When is it not an error?
    a.

A page fault is when a running program tries to access some virtual memory in its address space that is not mapped to physical memory. Page faults will also occur in other situations.

There are three types of Page Faults

Minor If there is no mapping yet for the page, but it is a valid address. This could be memory asked for by `sbrk(2)` but not written to yet meaning that the operating system can wait for the first write before allocating space. The OS simply makes the page, loads it into memory, and moves on.

Major If the mapping to the page is not in memory but on disk. What this will do is swap the page into memory and swap another page out. If this happens frequently enough, your program is said to *thrash* the MMU.

Invalid When you try to write to a non-writable memory address or read to a non-readable memory address. The MMU generates an invalid fault and the OS will usually generate a `SIGSEGV` meaning segmentation violation meaning that you wrote outside the segment that you could write to.

13. What is Spatial and Temporal Locality? Swapping? Swap file? Demand Paging?
    a.

Temporal locality: The concept that a resource that is referenced at one point in time will be referenced again sometime in the near future.

Spatial locality: The concept that likelihood of referencing a resource is higher if a resource near it was just referenced.

Swapping:

To replace pages or segments of data in memory. Swapping is a useful technique that enables a computer to execute programs and manipulate data files larger than main memory. The operating system copies as much data as possible into main memory, and leaves the rest on the disk. When the operating system needs data from the disk, it exchanges a portion of data (called a *page* or *segment*) in main memory with a portion of data on the disk.

DOS does not perform swapping, but most other operating systems, including OS/2, Windows, and UNIX, do.

Swapping is often called *paging*.

Swap File:

Alternatively referred to as a **page file** or **paging file**, a **swap file** is a file on a hard drive that is used as a temporary location to store information not being used by the computer RAM. By using a swap file, a computer can use more memory than what is physically installed in the computer. However, if the computer is low on drive space the computer can run slower because of the inability of the swap file to grow

Demand Paging:

In virtual memory systems, demand paging is a type of *swapping* in which pages of data are not copied from disk to RAM until they are needed. In contrast, some virtual memory systems use *anticipatory paging,* in which the operating system attempts to anticipate which data will be needed next and copies it to RAM before it is actually required.

3. Processes and Threads

14. What resources are shared between threads in the same process?
    a. Basically everything except that they have private register values and stack segments. They share Heap memory ,Global variables, Address space(page table).

15. Explain the operating system actions required to perform a process context switchd
    a.
    A process switch is a operating system scheduler change from one running program to another. This requires saving all of the state of the currently executing program, including its register state, associated kernel state, and all of its virtual memory configuration. All of the state of the new program is then loaded and execution continues.

16. Explain the actions required to perform a thread context switch (to a thread in the same process)
    a.
    A thread switch shifts from one thread to another, within one program. Threads within a program are full execution contexts, but they share one address space with other threads in the program. A thread switch is cheaper than a full context switch since the **memory management unit does not need to be reconfigured**.
    A context switch can informally mean either a process or thread switch, depending on the context (pun intended).

17. How can a process be orphaned? What does the process do about it?

Resource:
https://www.gmarik.info/blog/2012/orphan-vs-zombie-vs-daemon-processes/

An orphan process is a computer **process whose parent process has finished or terminated**, though it remains running itself.
In a Unix-like operating system any orphaned process will be immediately adopted by the special `init` system process. This operation is called re-parenting and occurs automatically. Even though technically the process has the `init` process as its parent, it is still called an orphan process since the process that originally created it no longer exists.

18. How do you create a process zombie?

On Unix and Unix-like computer operating systems, **a zombie process or defunct process is a process that has completed execution but still has an entry in the process table**. This entry is still needed to allow the parent process to read its child's exit status. The term zombie process derives from the common definition of zombie — an undead person. In the term's metaphor, the child process has "died" but has not yet been "reaped". Also, unlike normal processes, the kill command has no effect on a zombie process.

A **zombie process is not the same as an orphan process**. An orphan process is a process that is still executing, but whose parent has died. They do not become zombie processes; instead, they are adopted by `init` (process ID 1), which waits on its children.

19. Under what conditions will a multi-threaded process exit? (List at least 4)
    a. One of the threads call exit()
    b. The main thread calls return
    c. The process is terminated by the kill() system call.
    d. When you kill your computer… :-((help me come up more)
    e. Calling pthread_exit(0) in main thread <- don't think this is right
    f. A thread writes to address zero (segfault)

4. Scheduling

20. Define arrival time, pre-emption, turnaround time, waiting time and response time in the context of scheduling algorithms. What is starvation? Which scheduling policies have the possibility of resulting in starvation?
    a. `start_time` is the wall-clock start time of the process (CPU starts working on it) `end_time` is the end wall-clock of the process (CPU finishes the process) `run_time` is the total amount of CPU time required

b. `arrival_time` is the time the process enters the scheduler (CPU may not start working on it)
c. `turnaround_time = end_tim fe - arrival_time`
d. `response_time = start_time - arrival_time`
e. `wait_time = (end_time - arrival_time) - run_time`
f. Without preemption processes will run until they are unable to utilize the CPU any further. For example the following conditions would remove a process from the CPU and the CPU would be available to be scheduled for other processes: The process terminates due to a signal, is blocked waiting for concurrency primitive, or exits normally. Thus once a process is scheduled it will continue even if another process with a high priority (e.g. shorter job) appears on the ready queue. Preemptive SJF would also suffer starvation.

Starvation happens when a job is constantly pushed off because there are higher priority jobs keeping running. Any scheduler that uses a form of prioritization can result in starvation because earlier processes may never be scheduled to run (assigned a CPU).

21. Which scheduling algorithm results the smallest average wait time?
    a. SJF

22. What scheduling algorithm has the longest average response time? shortest total wait time?
    a. FCFS/SJF (need reasoning)

23. Describe Round-Robin scheduling and its performance advantages and disadvantages

• Solution to fairness and starvation
- Preempt job after some time slice or quantum
- When preempted, move to back of FIFO queue
- (Most systems do some flavor of this)

• Advantages:
- Fair allocation of CPU across jobs
- Low average waiting time when job lengths vary
- Good for responsiveness if small number or jobs

• Disadvantages?
Bad average completion time for similar-sized job
Overhead due to context switch?

24. Describe the First Come First Serve (FCFS) scheduling algorithm. Explain how it leads to the convoy effect.

    a. FCFS simply queues processes in the order that they arrive in the ready queue.

    http://www.geeksforgeeks.org/convoy-effect-operating-systems/

What is the Convoy Effect?

"The Convoy Effect is where I/O intensive processes are continually backed up, waiting for CPU-intensive processes that hog the CPU. This results in poor I/O performance, even for processes that have tiny CPU needs."

Suppose the CPU is currently assigned to a CPU intensive task and there is a set of I/O intensive processes that are in the ready queue. These processes require just a tiny amount of CPU time but they are unable to proceed because they are waiting for the CPU-intensive task to be removed from the processor. These processes are starved until the the CPU bound process releases the CPU. But the CPU will rarely be released (for example in the case of a FCFS scheduler, we must wait until the processes is blocked due to an I/O request). The I/O intensive processes can now finally satisfy their CPU needs, which they can do quickly because their CPU needs are small and the CPU is assigned back to the CPU-intensive process again. Thus the I/O performance of the whole system suffers through an indirect effect of starvation of CPU needs of all processes.

This effect is usually discussed in the context of FCFS scheduler, however a round robin scheduler can also exhibit the Convoy effect for long time-quanta.

25. Describe the Pre-emptive and Non-preemptive SJF scheduling algorithms.
    a. Preemptive scheduling algorithms allow the scheduler to put down the job it's working on and allow job to jump in if the new job comes in with a shorter runtime than **the total runtime** of the current job. Nonpreemptive won't do this.

26. How does the length of the time quantum affect Round-Robin scheduling? What is the problem if the quantum is too small? In the limit of large time slices Round Robin is identical to _____?

    a. If the quantum is too small, the CPU wastes a lot of time in context switching.
       Limit of large time slices RR is identical to FCFS.

27. What reasons might cause a scheduler switch a process from the running to the ready state?
    a. Preemptive scheduler find a job with a higher priority

5. Synchronization and Deadlock

28. Define circular wait, mutual exclusion, hold and wait, and no-preemption. How are these related to deadlock?

    Mutual Exclusion: The resource cannot be shared

    Circular Wait: There exists a cycle in the Resource Allocation Graph. There exists a set of processes {P1,P2,...} such that P1 is waiting for resources held by P2, which is banwaiting for P3,..., which is waiting for P1.

    Hold and Wait: A process acquires an incomplete set of resources and holds onto them while waiting for the other resources.

    No pre-emption: Once a process has acquired a resource, the resource cannot be taken away from a process and the process will not voluntarily give up a resource.

29. What problem does the Banker's Algorithm solve?
    a. circular wait

30. What is the difference between Deadlock Prevention, Deadlock Detection and Deadlock Avoidance?
    a. https://stackoverflow.com/questions/2485608/what-the-difference-between-deadlock-avoidance-and-deadlock-prevention
    b. Avoid: Don't share resources across processes / mulitple threads
    c. Prevent: When accsessing shared resources, use a semaphore. If locking multiple semaphores, be sure to unlock in the reverse order of locking. Always be sure to handle errors within the critical sections so the semaphore is released under all conditions.
    d. Deadlock prevention is making sure that deadlock cannot happen, meaning that you break a coffman condition. This works the best inside a single program and the

software engineer making the choice to break a certain coffman condition. Consider the Banker's Algorithm. It is another algorithm for deadlock avoidance. The whole implementation is outside the scope of this class, just know that there are more generalized algorithms for operating systems.

e. Deadlock detection on the other hand is allowing the system to enter a deadlocked state. After entering, the system uses the information that it has to break deadlock. As an example, consider multiple processes accessing files. The operating system is able to keep track of all of the files/resources through file descriptors at some level (either abstracted through an API or directly). If the operating system detects a directed cycle in the operating system file descriptor table it may break one process' hold (through scheduling for example) and let the system proceed.

31. Sketch how to use condition-variable based barrier to ensure your main game loop does not start until the audio and graphic threads have initialized the hardware and are ready.

```
barrier_t barrier;
barrier_init(&barrier, 3); // 3 threads for audio/ graphic/ main loop
pthread_create()
pthread_create();  2 threads for audio/graphic, use barrier_wait when they've initialized the hardware and are ready
barrier_wait();
while(1) {
        // main loop
}
```

32. Implement a producer-consumer fixed sized array using condition variables and mutex lock.

https://docs.oracle.com/cd/E19455-01/806-5257/sync-31/index.html

33. Create an incorrect solution to the CSP for 2 processes that breaks: i) Mutual exclusion. ii) Bounded wait.

```
\\ Candidate #4
raise my flag
if your flag is raised, wait until my turn
// Do Critical Section stuff
turn = yourid
lower my flag
```

34. Create a reader-writer implementation that suffers from a subtle problem. Explain your subtle bug.
    a. read() {
       lock(&m)
       // do read stuff
       unlock(&m)
       }
    b. write() {
       lock(&m)
       // do write stuff
       unlock(&m)
       }
    c. One reader reads will block other readers

6. IPC and signals

35. Write brief code to redirect future standard output to a file.
    a. dup2(fd, STDOUT_FILENO)

36. Write a brief code example that uses dup2 and fork to redirect a child process output to a pipe

```
int pipe_[2];
pipe(pipe_);
pid_t child = fork();
if(child == 0) {
        dup2(pipe_[1], STDOUT_FILENO);
        // do stuff
}
```

37. Give an example of kernel generated signal. List 2 calls that can a process can use to generate a SIGUSR1.
    a. SIGCHLD
    b. raise(), kill()

38. What signals can be caught or ignored?
    a. All but SIGKILL and SIGSTOP (I'm not sure)(You are right)

39. What signals cannot be caught? What is signal disposition?

    SIGKILL and SIGSTOP cannot be caught/ignored

    a. For each process, each signal has a disposition which means what action will occur when a signal is delivered to the process. For example, the default disposition SIGINT is to terminate it. The signal disposition can be changed by calling signal() (which is simple but not portable as there are subtle variations in its implementation on different POSIX architectures and also not recommended for multi-threaded programs) or `sigaction` (discussed later). You can imagine the processes' disposition to all possible signals as a table of function pointers entries (one for each possible signal).

       The default disposition for signals can be to ignore the signal, stop the process, continue a stopped process, terminate the process, or terminate the process and also dump a 'core' file. Note a core file is a representation of the processes' memory state that can be inspected using a debugger.

40. Write code that uses sigaction and a signal set to create a SIGALRM handler.

```
struct sigaction sa;
sigemptyset(&sa.sa_mask);
sa.sa_handler = handler;
sa.sa_flag = 0;
sigaction(SIGALRM, &sa, NULL);
```

```
signal(SIGALRM, handler);
```

41. Why is it unsafe to call printf, and malloc inside a signal handler?

    a. Because they're not *reentrant* (image signal handler is invoked in the middle of printf)

       Printf uses a internal static buffer/ malloc manages a free (link) list.

7. Networking

42. Explain the purpose of socket, bind, listen accept functions

    Socket: set up a telephone device

    Bind: bind to a telephone number so that others can call me.

Listen: allow others to call my number. If busy, make a backlog and accept her call later.

Accept: pick up the phone and start communication.

^Genius

43. Write brief (single-threaded) code using getaddrinfo to create a UDP IPv4 server. Your server should print the contents of the packet or stream to standard out until an exclamation point "!" is read.

```
int main(int argc, char **argv)
{
    int s;
    struct addrinfo hints, *res;
    memset(&hints, 0, sizeof(hints));
    hints.ai_family = AF_INET;
    hints.ai_socktype =  SOCK_DGRAM;
    hints.ai_flags =  AI_PASSIVE;
    getaddrinfo(NULL, "1234", &hints, &res);
    int sockfd = socket(res->ai_family, res->ai_socktype, res->ai_protocol);
    if (bind(sockfd, res->ai_addr, res->ai_addrlen) != 0) {
        perror("bind()");
        exit(1);
    }
    //struct sockaddr_storage addr;
    //int addrlen = sizeof(addr);
    char c = 0;
    while(c != '!'){
        read(sockfd, &(char*)c, 1);
        printf("%c", c);
    }
    close(sockfd);
    return 0;
}
```

44. Write brief (single-threaded) code using getaddrinfo to create a TCP IPv4 server. Your server should print the contents of the packet or stream to standard out until an exclamation point "!" is read.

```
int main(int argc, char **argv)
{
    int s;
    int sock_fd = socket(AF_INET, SOCK_STREAM, 0);
    struct addrinfo hints, *result;
    memset(&hints, 0, sizeof(struct addrinfo));
    hints.ai_family = AF_INET;
    hints.ai_socktype = SOCK_STREAM;
    hints.ai_flags = AI_PASSIVE;
    s = getaddrinfo(NULL, "1234", &hints, &result);
    if (s != 0) {
            fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(s));
            exit(1);
    }
    if (bind(sock_fd, result->ai_addr, result->ai_addrlen) != 0) {
        perror("bind()");
        exit(1);
    }
    if (listen(sock_fd, 10) != 0) {
        perror("listen()");
        exit(1);
    }
    int client_fd = accept(sock_fd, NULL, NULL);
    char c = 0;
    while(c != '!') {
        read(client_fd, &(char*)c, 1);
        printf("%c", c);
    }
    close(client_fd);
    close(sock_fd);
    return 0;
}
```

45. Explain the main differences between using select and epoll. What are edge- and level-triggered epoll modes?
    a. Select is of complexity O(n); epoll is O(1)

ET reports an event when it's available once.
LT reports an event as long as it's still available.

46. Describe the services provided by TCP but not UDP.

TCP will automatically manage resending packets, ignoring duplicate packets, re-arranging out-of-order packets and changing the rate at which packets are sent.

47. How does TCP connection establishment work? And h*ow does it affect latency in HTTP1.0 vs HTTP1.1?
    a. Three-way handshake TCP Handshake
    b. In HTTP1.1, all connections are persistent unless declared otherwise. Hence it improve the latency for later communication after initialization.

48. Wrap a version of read in a loop to read up to 16KB into a buffer from a pipe or socket. Handle restarts (EINTR), and socket-closed events (return 0).

```
size_t curr = 0;
ssize_t read_bytes;
char buffer[16 * 1024];
while((read_bytes = read(sockfd, buffer + curr, 16 * 1024 - curr) != 0) {
        if(read_bytes == -1) {
                if(errno == EINTR)
                        continue;
                exit(1);
        }
        curr += read_bytes;
        if(curr > total)
                break;
}
```

49. How is Domain Name System (DNS) related to IP and UDP? When does host resolution not cause traffic?
    a. A system called "DNS" (Domain Name Service) is used to convert website address to IP address. If a machine does not hold the answer locally then it sends a UDP packet to a local DNS server. This server in turn may query other upstream DNS servers.

b. When local cache stores the data (host name to ip address)

50. What is NAT and where and why is it used?

https://en.wikipedia.org/wiki/Network_address_translation
Network Address Translation allows a single device, such as a router, to act as an agent between the Internet (or "public network") and a local (or "private") network. This means that only a single, unique IP address is required to represent an entire group of computers.

8. Files

51. Write code that uses fseek, ftell, read and write to copy the second half of the contents of a file to a pipe.

```
fseek(file, 0, SEEK_END);
len = ftell(file);
fseek(file, len/2, SEEK_SET);
Buffer[1024];
Int read_bytes;
while((read_bytes = read(file, buffer, 1024)) != 0) {
        write(pipe[1], buffer, read_bytes);
}
```

52. Write code that uses open, fstat, mmap to print in reverse the contents of a file to stderr.

```
struct stat sb;
int fd = open("pathname", O_RD_ONLY, 0400);
fstat(fd, &sb);
Size_t size= sb.st_size;
void *addr = mmap(0, size, PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);
for(size_t i = 0; i < size; ++i) {
    printf(“%c”, *(addr+size-i));
}
```

53. Write brief code to create a symbolic link and hard link to the file /etc/password

```
symlink(“/etc/password”, “s_link”);
link(“/etc/password”,  “h_link”);
```

54. "Creating a symlink in my home directory to the file /secret.txt succeeds but creating a hard link fails" Why?
    a. /secret.txt is on a different filesystem

55. Briefly explain permission bits (including sticky and setuid bits) for files and directories.

https://en.wikipedia.org/wiki/Setuid

setuid and **setgid** (short for "**set u**ser **ID** upon execution" and "**set g**roup **ID** upon execution", respectively)[1] are Unix access rights flags that allow users to run an executablewith the permissions of the executable's owner or group respectively and to change behaviour in directories. They are often used to allow users on a computer system to run programs with temporarily elevated privileges in order to perform a specific task. While the assumed user id or group id privileges provided are not always elevated, at a minimum they are specific.

```
The restricted deletion flag or sticky bit is a single bit, whose interpretation depends on the file type. For directories, it prevents unprivileged users from removing or renaming a file in the directory unless they own the file or the directory; this is called the restricted deletion flag for the directory, and is commonly found on world-writable directories like /tmp. For regular files on some older systems, the bit saves the program's text image on the swap device so it will load more quickly when run; this is called the sticky bit.
```

56. Write brief code to create a function that returns true (1) only if a path is a directory.

```
lstat(pathname, &sb);
if(S_ISDIR(sb.st_mode)) {
        Return 1;
}dp
```

57. Write brief code to recursive search user's home directory and sub-directories (use getenv) for a file named "xkcd-functional.png' If the file is found, print the full path to stdout.

```
int search(char* path, char* filename, char** filepath) {
      DIR* dir = opendir (path);
      if (dir == NULL) {
            perror ("opendir");
            return 0;
```

```
	}
	struct dirent* de;
	while ((de = readdir (dir)) != NULL) {
		if (strcmp (de->d_name, ".") == 0 || strcmp (de->d_name, "..")
== 0) continue;
		char* newpath = malloc (strlen (path) + strlen (de->d_name) + 2);
		sprintf (newpath, "%s/%s", path, de->d_name);
		if (strcmp (de->d_name, filename) == 0) {
			*filepath = newpath;
			closedir (dir);
			return 1;
		}
		struct stat s;
		if (lstat (newpath, &s) == 0 && !S_ISLNK (s.st_mode) && S_ISDIR
		(s.st_mode)) {
			if (search (newpath, filename, filepath) == 1) {
				closedir (dir);
				free (newpath);
				return 1;
			}
		}
		free (newpath);
	}
	closedir (dir);
	return 0;
}
int main(int argc, char * argv[]) {
	char* path=getenv("HOME");
	char* filepath;
	search(path, “xkcd-functional.png”, &filepath);
	printf(filepath);
}
```

58. The file 'installmeplz' can't be run (it's owned by root and is not executable). Explain how to use sudo, chown and chmod shell commands, to change the ownership to you and ensure that it is executable.
    a. sudo chown user installmeplz
       chmod +x installmeplz
       ./installmeplz

9. Filesystem

Assume 10 direct blocks, a pointer to an indirect block, double-indirect, and triple indirect block, and block size 4KB.

59. A file uses 10 direct blocks, a completely full indirect block and one double-indirect block. The latter has just one entry to a half-full indirect block. How many disk blocks does the file use, including its content, and all indirect, double-indirect blocks, but not the inode itself? A sketch would be useful.
    (10 * 4KB +
    4KB + 1024 * 4KB +
    4KB + 4KB + 512 * 4KB) / 4KB = 1549 disk blocks

60. How many i-node reads are required to fetch the file access time at /var/log/dmesg? Assume the inode of (/) is cached in memory. Would your answer change if the file was created as a symbolic link? Hard link?
    a. 3?

61. What information is stored in an i-node? What file system information is not?
    a. Pointer to data block, protection, timestamp, size, signature pointer
    b. Filename is not stored in an i-node

62. Using a version of stat, write code to determine a file's size and return -1 if the file does not exist, return -2 if the file is a directory or -3 if it is a symbolic link.

    If(stat(pathname, &sb) == -1)

```
        Return -1;
if(S_ISDIR(sb.st_mode))
        Return -2;
if(S_ISLNK(sb.st_mode))
        Return -3;
```

63. If an i-node based file uses 10 direct and n single-indirect blocks (1<= n <=1024), what is the smallest and largest that the file contents can be in bytes? You can leave your answer as an expression.

    Smallest: 0KB (???) 4KB*10+1byte ?
    Largest: 10 * 4KB +
    4KB + n * 4KB

64. When would fstat(open(path,O_RDONLY),&s) return different information in s than lstat(path,&s)?
    a. When pathname is a symlink, fstat returns info of the file to which the symlink refers, while lstat returns info about the link itself.