QUIZ 4

TOTAL POINTS 100

TRUE/FALSE QUESTIONS – 1PT EACH [30 PTS]

1. **[True]** Shared memory IPC comes with built-in (kernel provided) synchronization
2. **[True]** FIFOs persist without any processes connected to them
3. **[False]** Shared memory and memory mapped files require 3 times memory overhead compared to FIFO and MQ
4. **[True]** Pipes are supported by a First-In-First-Out bounded buffer given by the Kernel
5. **[True]** POSIX message queues support separate priority levels for the messages
6. **[True]** In POSIX message queues, the order of the messages is always FIFO without any exception
7. **[False]** A unnamed pipe can be established only between processes in the same family tree
8. **[True]** A unnamed pipe does not exist without processes connected to both ends
9. **[True]** In POSIX message queue, you can configure message size and number of messages
10. **[True]** In shared memory IPC, the Virtual Memory manager maps the same piece of physical memory to the address space of each sharing process
11. **[True]** After creating a shared memory segment with shm_open() function, the default size of the segment is 0
12. **[True]** POSIX IPC objects (message queues, shared memory, semaphores) can be found under **/dev/mqueue** and **/dev/shm** directories
13. **[True]** You can set/change the length of the shared memory segment using **ftruncate()** function
14. **[True]** In POSIX, names for message queue, shared memory and kernel semaphores must start with a "/"
15. **[True] sem_unlink()** function permanently removes a semaphore from the kernel
16. **[True]** You must use **ftruncate()** before using a shared memory segment
17. **[False]** You must call **mmap()** before using (i.e., read/write) a shared memory segment
18. **[True]** UDP protocol deals with retransmitting packets in case they are lost in route
19. **[True]** TCP protocol is more heavy-weight because it maintains state information about the connection
20. **[True]** Routing protocols are dynamic: they reconfigure under network topology change or outage automatically
21. **[True]** Domain Naming System (DNS) can be used for load balancing and faster content delivery
22. **[False]** The ping command is used to test whether you can make TCP connections with a remote host
23. **[False]** The accept() function needs to be called the same number of times as the number of client-side connect() calls to accept all of them
24. **[True]** HTTP protocol uses TCP underneath
25. **[True]** Ports [0,1023] are reserved for well-known services (e.g., HTTP, SMTP, DNS, TELNET)
26. **[True]** The master socket in a TCP server is used only to accept connections, not to run conversations with clients
27. **[False]** A socket is a pair of IP-address and port number combination in both client and server side
28. **[True]** For making a socket on the client side, the port number is chosen by the OS randomly from the available pool
29. **[True]** A socket is like other file descriptors and is added to the Descriptor Table
30. **[True]** UDP is connectionless while TCP is connection oriented

FILE SYSTEMS

31. [20 pts] Assume that a file system has each disk block of size 4KB and each block pointer of 4 bytes. In addition, the each inode in this system has 14 direct pointers, 2 single indirect pointers, 1 double indirect and 1 triple indirect pointer. Ignoring the space for inode, answer the following questions for this file system:

(a) What is the maximum possible file size?

$$No\ of\ addresses\ per\ block = \frac{4KB}{4\ bytes} = 2^{10}$$
$$Maximum\ File\ Size = (14\ Direct + 2\ Sinlge + 1\ Double + 1\ Triple) * 4KB$$
$$= (14 + 2 * 2^{10} + 2^{20} + 2^{30}) * 4 * 2^{10} = 4100GB$$

(b) How much overhead (amount of non-data information) for the maximum file size derived in (a)?

$$Amount\ of\ nondata\ info = (14 + 2 * 2^{10} + 2^{20} + 2^{30}) * 4 = 4GB$$

(c) How much overhead for a file of size 6GB?

$$Amount\ of\ nondata\ info = \frac{6GB}{4KB} * 4 = \frac{6 * 2^{30}}{4 * 2^{10}} * 4 = 6 * 2^{20} = 6MB$$

SIGNALS

32. [10 pts] The following code will create a Zombie child process because the child process is terminated and the parent process is busy in a loop without calling `wait()` function. Now, modify this program by handling `SIGCHLD` signal so that no Zombie process is created. The parent process cannot call `wait()` directly in the main(). However, calling wait() from inside the signal handler is fine. The main still must go to the infinite while loop. You can add helper functions.

```
int main(){
    if (fork()== 0) // child process
        exit(0);
    else // parent process
        signal(SIGCHILD,SIG_IGN);
        while (true);
}
```

Explanation: if we call the 'signal(SIGCHLD,SIG_IGN)', then the SIGCHLD signal is ignored by the system, and the child process entry is deleted from the process table. Thus, no zombie is created.

33. [15 pts] Consider the program below and answer the following questions with proper explanation.

a) What is the output? How much time does the program take to run?[10 points]

b) What is the output with line 5 commented? How much time will it take now? [10 points]

```
1 void signal_handler (int signo){
2   printf ("Got SIGUSR1\n");
3 }
4 int main (){
5   signal (SIGUSR1, signal_handler); //comment out for b)
6   int pid = fork ();
7   if (pid == 0){// chilld process
8       for (int i=0; i<5; i++){
9           kill(getppid(), SIGUSR1);
10          sleep (1);
11      }
12 }else{ // parent process
13      wait(0);
14 }
15}
```

a) The output is displayed as five lines of "Got SIGUSR1". It takes the program around 5 seconds to run.

b) With line 5 commented, the output is "User defined signal 1" in one line. It takes the program around 5 seconds to run.

34. **[15 pts]** Write a wrapper class KernelSemaphore on top of POSIX kernel semaphore. See sem_overview(7) in man pages or linux.die.net to learn about kernel semaphores. Test your KernelSemaphore class by by setting the initial value to 0. Then write 2 programs – one waits for the semaphore and the other one releases (i.e., V()') it. The header for the KernelSemaphore and the 2 programs in questions are provided in the below. You should make sure that the consumer program can only print out its prompt after the producer program has released the semaphore.

```cpp
class KernelSemaphore{
    string name;
public:
    KernelSemaphore (string _name, int _init_value);
    void P();
    void V();
    ~KernelSemaphore ();
};
// producer.cpp (Run this first in a terminal)
int main (){
    cout << "This program will create the semaphore, initialize it to 0, ";
    cout << "then produce some data and finally V() the semaphore" << endl;
    KernelSemaphore ks ("/my_kernel_sema", 0);
    sleep (rand () % 10); // sleep a random amount of seconds
    ks.V();
}
// consumer.cpp (Run this second in another terminal)
int main (){
    KernelSemaphore ks ("/my_kernel_sema", 0);
    ks.P();
    cout << "I can tell the producer is done"<< endl;
}
```

35. **[10 pts]** Assume a very computer system from a company that used a very old legacy device whose path is /dev/legacy/specialdevice and it is about to be decommissioned. However, there are several important pieces of software who use this legacy device to log their output and you cannot change those. That means, the path /dev/legacy/specialdevice must continue to exist although the underlying physical device must be replaced. Now, what can you do to make sure all legacy tools and software continues running w/o problem without putting a new physical device in the above path? Note: you may forward all traffic to the legacy device to let's say /sys/logfile path.

To make sure all legacy tools and software continues running without problem, we need to link the logfile with the directory of special device by using symbolic link. The association of a new file with the path to the original file will make sure removing the symlink only removes the pointer to the original file, but not the file itself. Thus, we are still able to use the legacy tools and software in the old device even if it's decommissioned. The command line to implement this "ln -s /dev/legacy/specialdevice /sys/logfile".