CSCE 313 # QUIZ 4 SPRING 2020

TOTAL POINTS 100

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

1. Shared memory IPC comes with built-in (kernel provided) synchronization
   False
2. FIFOs persist without any processes connected to them
   True
3. Shared memory and memory mapped files require 3 times memory overhead compared to FIFO and MQ
   False
4. Pipes are supported by a First-In-First-Out bounded buffer given by the Kernel
   True
5. POSIX message queues support separate priority levels for the messages
   True
6. In POSIX message queues, the order of the messages is always FIFO without any exception
   False
7. A unnamed pipe can be established only between processes in the same family tree
   True
8. A unnamed pipe does not exist without processes connected to both ends
   True
9. In POSIX message queue, you can configure message size and number of messages
   True
10. In shared memory IPC, the Virtual Memory manager maps the same piece of physical memory to the address space of each sharing process
    True
11. After creating a shared memory segment with shm_open() function, the default size of the segment is 0
    True
12. POSIX IPC objects (message queues, shared memory, semaphores) can be found under **/dev/mqueue** and **/dev/shm** directories
    True
13. You can set/change the length of the shared memory segment using **ftruncate()** function
    True
14. In POSIX, names for message queue, shared memory and kernel semaphores must start with a "/"
    True
15. **sem_unlink()** function permanently removes a semaphore from the kernel
    True
16. You must use **ftruncate()** before using a shared memory segment
    True
17. You must call **mmap()** before using (i.e., read/write) a shared memory segment
    True
18. UDP protocol deals with retransmitting packets in case they are lost in route
    True
19. TCP protocol is more heavy-weight because it maintains state information about the connection
    True
20. Routing protocols are dynamic: they reconfigure under network topology change or outage automatically
    True
21. Domain Naming System (DNS) can be used for load balancing and faster content delivery

True

22. The ping command is used to test whether you can make TCP connections with a remote host
    False
23. 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
    True
24. HTTP protocol uses TCP underneath
    True
25. Ports [0,1023] are reserved for well-known services (e.g., HTTP, SMTP, DNS, TELNET)
    True
26. The master socket in a TCP server is used only to accept connections, not to run conversations with clients
    True
27. A socket is a pair of IP-address and port number combination in both client and server side
    True
28. For making a socket on the client side, the port number is chosen by the OS randomly from the available pool
    True
29. A socket is like other file descriptors and is added to the Descriptor Table
    True
30. UDP is connectionless while TCP is connection oriented
    True

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?
    Size of direct pointer = 14
    Size of indirect pointer = 2 ^ 10
    Size of double indirect pointer = 2 ^ 20
    Size of tribble indirect pointer = 2^30
    Maximum Size = 4*(14 + 2* 2^10+ 2^20 + 2^30) = 56KB + 8MB + 4GB + 4TB = 4 * 1074792462 (KB)
    (b) How much overhead (amount of non-data information) for the maximum file size derived in (a)?
    Overhead = 4*(0 + 2*(2^10)+ (2^10 + 2^20) + (2^10+2^20+2^30)
    = 8KB + 4KB(1+2^10) + 4KB(1+2^10+2^20)
    (c) How much overhead for a file of size 6GB?
    If we counting the direct pointer as overhead:
    OverHead=(6*(2^30))/(4*(2^10)) * 4 (Byte)
    If we not counting the direct pointer as overhead:
    OverHead=(6*(2^30))/(4*(2^10)) * 4 – (14 *4) Byte

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.

```
void signal_handler(){
    wait(0);
```

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

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]

Got SIGUSR1
Got SIGUSR1
Got SIGUSR1
Got SIGUSR1
Got SIGUSR1

It takes at least 5 seconds to run,because each iteration of child in the for loop takes 1 seconds, and there are 5 iteration. The 5 lines of output because each time the iteration complete , the kill function send the SIGUSR1 signal to the parent function. Thus, parent function will execute the signal handler function.

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

No output

At least 5 seconds to finish

No output is because we do not specify what does the program behave if it receive the SIGUSR1 signal, thus it will do nothing.

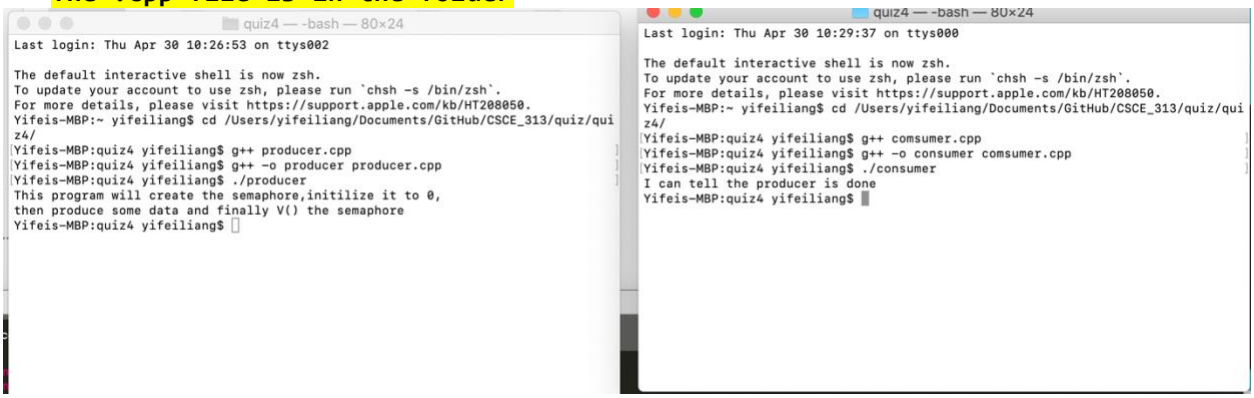
```
1  void signal_handler (int signo){
2    printf ("Got SIGUSR1\n");
3  }
4  int main (){
5    signal (SIGUSR1, signal_handler); //comment out for b)
6    int pid = fork ();
7    if (pid == 0){// chilld process
8          for (int i=0; i<5; i++){
9                kill(getppid(), SIGUSR1);
10      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

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.

```
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;
}
```

The .cpp file is in the folder



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 decomissioned. 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.

Suppose we forward all the traffic from the legacy device to /sys/logfile. We can make a hard link to link every file in/dev/legacy/specialdevide to /sys/logfile. Thus even if we delete all the file in

/dev/legacy/specialdeivce, the link in the /sys/logfile will not be affected and will functioning properly as deleted one.