Name: Matthew Mendoza

1. Including the initial parent process, how many processes are created by the program shown in Figure 3.31 in your textbook?

   Eight processes are created

2. When a process creates a new process using the fork() operation, which of the following state is shared between the parent process and the child process?

   a. Stack
   b. Heap
   c. Shared memory segments

3. Describe the actions taken by a kernel to context-switch between processes.

By saving the value of the CPU registers from the tread and is then switched out and restoring the registers in the CPU with a new thread scheduled.

4. What is the difference between ordinary pipes and named pipes?

Ordinary pipes are unidirectional, allowing only one-way communication; in contrast, named pipes allow bidirectional communication, and no parent–child relationship is required.

5. Give an example of a situation in which ordinary pipes are more suitable than named pipes and an example of a situation in which named pipes are more suitable than ordinary pipes.

   a. **Ordinary pipes are more suitable than named pipes**

we want to establish communication between two specific processes on the same machine, then using ordinary pipes is more suitable than using named pipes because named pipes involve more overhead in this situation

   b. **Named pipes are more suitable than ordinary pipes**

A typical scenario, a named pipe has several writers. Additionally, named pipes continue to exist after communicating processes have finished.

6. What are the benefits and the disadvantages of Synchronous and asynchronous communication? (Consider both the system level and the programmer level.)

A benefit of synchronous communication is that it allows a rendezvous between the sender and receiver. A disadvantage of a blocking send is that a rendezvous may not be required and the message could be delivered asynchronously. As a result, message-passing systems often provide both forms of synchronization.

7. What are the outputs of the following code? Please test them on Linux
   Fork system call use for creates a new process, which is called ***child process***, which runs concurrently with process (which process called system call fork) and this process is called ***parent process***. After a new child process created, both processes will execute the next instruction following the fork() system call. A child process uses the same pc(program counter), same CPU registers, same open files which use in the parent process.

```
int main(){
    fork();
    fprintf(stderr, "Hello/n");
}
```

//print out two Hello lines

```
int main(){
    fork();
    fork();
    fprintf(stderr, "Hello/n");
}
```

//print out four Hello lines

```
int main(){
    fork();
    fork();
    fork();
    fprintf(stderr, "Hello/n");
}
```

//print out eight Hello lines

There are 3 fork calls. Each fork call creates a new process. These processes have a parent and a child. Each call will create another layer: it is 2^n processes for fork() calls.

8. What is a zombie? Why does the zombie problem arise? Why must wait() be called on children?

A zombie is a parent process that has been terminated but has not invoked **wait()**. The process will still be in the process table. The zombie problem arises because not all processes are terminated when **exit()** is called. **wait()** must be called before entries are released.

9. What is the difference between an orphan and zombie processes?

A zombie is a parent process that has terminated but has not invoked the wait() method; in contrast, an orphan is a child process of the zombie.

10. What is a thread?

A thread is a basic unit of CPU utilization; it comprises a thread ID, a program counter (PC), a register set, and a stack. It shares with other threads belonging to the same process its code section, data section, and other operating-system resources, such as open files and signals. A traditional process has a single thread of control. If a process has multiple threads of control, it can perform more than one task at a time.

11. What is the difference between a thread and a process?

A thread is a path of execution within a process. A process can contain multiple threads.

12. What resources are used when a thread created? How do they differ from those when a process is created?

Since a thread is a part of the process, no additional resources are used when a thread is created, instead, it shares the memory space of the process from which this particular thread has been created.

13. What is multithreading? What are its benefits?

Multithreading allows the execution of multiple parts of a program at the same time. On a multiprocessor system, multiple threads can concurrently run on multiple CPUs. Therefore, multithreaded programs can run much faster than on a uniprocessor system. They can also be faster than a program using multiple processes, because threads require fewer resources and generate less overhead.

14. In a multi-threaded process, if one thread is busy on I/O will the entire process be blocked?

If the process has only one thread, then yes. If the process has multiple threads, then normally no if the operating system supports multithreading

15. What is better (why and how): multi processes OR single process with multiple threads?

Multi processes: If you think in the analogy where multi processes are “cities” and threads are the “roads” a city where a lot of work can be done locally versus a small city sending out jobs and later receive materials to work with.