1.(15 Points) You are an OS developer. Your system is running multiple processes. Describe what happens when a process moves from the "Running" state to the "Blocked" state. Provide an example scenario where this transition occurs. Explain what might cause a process to move from the "Blocked" state back to the "Ready" state. Provide a specific example

When a process moves from Running state to the Blocked states indicates that a process cannot continue it's execution until a certain condition criteria is fulfilled to resume the process. Could happen when a process has to wait for resources. Once the I/O is complete the process will be moved back to the ready state.
<u>For example a browser loads up a page</u> into the volatile memory and allocates its process to a certain thread and puts it into the "Ready" state. Next, whenever the PCB executes this process, it puts it into the "Running" state and then the "Waiting"("Blocked") state for the operation to complete. Then, it is placed back into the "Ready" state for the processor to allocate it to the next process.

2.(15 Points) You are developing a multi-threaded application to simulate a factory production line. In
this simulation, there are multiple producer threads and multiple consumer threads. Producers generate parts and place them in a shared buffer, while consumers take parts from the buffer to assemble products. The shared buffer has a limited capacity and is implemented using shared memory. During testing, several issues have arisen, impacting the application's performance and reliability. Identify and discuss the potential problems associated with using shared memory space in the context of the producer-consumer problem. What problems could occur due to concurrent access to the shared memory buffer by multiple producer and consumer threads? Consider issues such as race conditions, data corruption, and inconsistent states.

Since we are simulating a factory production line we have a fixed buffer that needs to be continuously freed up in order for the producer to be able to place the factory items inside of the buffer. Moreover, since we are using a shared buffer, each thread has to wait for the active thread to finish its process before accessing the buffer. Whenever threads enter a race condition they affect the execution timing causing unpredictable errors and memory corruption. Whenever two or more threads access the same memory address they may rewrite it at the time without proper synchronization causing data inconsistency while modifying the file.

3.(15 Points) You are working on a multi-threaded application where different processes need to communicate with each other frequently. To facilitate this communication, you have two options: using blocking IPC or non-blocking IPC. Each method has its own advantages and disadvantages, and
the choice between them depends on the specific requirements of your application.

a. Define blocking IPC and non-blocking IPC.
Blocking IPC is a type of IPC that manages send and receive synchronously, meaning that while an operation is active the other operation is waiting for it to complete.

Non-blocking IPC, on the other hand, works asynchronously, without waiting for one another, causing result inconsistency.

b. Provide at least one examples of situations where blocking IPC is typically used.

A consumer and a producer case is a great example of a blocking IPC, whenever the shared memory is full the producer waits until the memory frees up by the consumer. This guarantees that the message is received properly by the producer, and no loss occurs. If the loss occurs, the producer resends the data, until it reaches the consumer successfully.

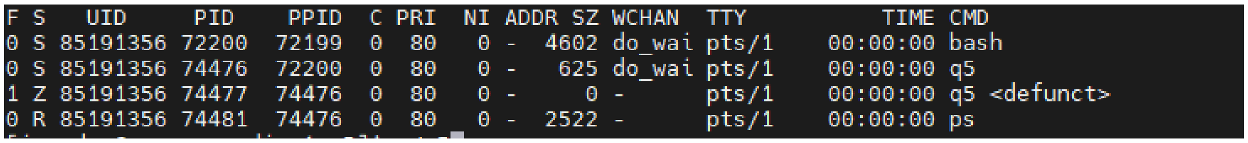
c. Provide at least one examples of situations where non-blocking IPC is typically used.

A messaging application is another great example illustrating non-blocking IPC, whenever a server puts a message inside of a buffer, the user doesn't always read the data off of the buffer. The server will continuously send messages disregarding whether the message is received successfully or not. This ensures that the server operates responsively and quickly enough to maintain connections.

4.(15 Points) For each of the scenarios below, identify which one is relates to concurrency and which to
parallelism.
a. When encoding a video, the task can be split into multiple parts (such as different frames or segments of the video). Each part can be processed simultaneously on different CPU cores, significantly speeding up the encoding process.
Parallelism because the process of different frames and segments of a video are occurring simultaneously on different CPU cores.
b. A web server can handle multiple client requests. It might switch between handling different requests rapidly, giving the appearance that all requests are being handled simultaneously
Concurrency since the web server is switching between different requests and performs them sequentially instead of doing them at the same time.

5.(20 Points) Write a C program that forks a child process that ultimately becomes a zombie process.
This zombie process must remain in the system for at least 10 seconds. Process states can be obtained
from the command " ps -l" The process states are shown below the S column; processes with a state of Z are zombies. The process identifier (PID) of the child process is listed in the PID column, and that of the parent is listed in the PPID column. Perhaps the easiest way to

determine that the child process is indeed a zombie is to run the program that you have written in the background (using the &) and then run
the command ps -l to determine whether the child is a zombie process. Because you do not want
too many zombie processes existing in the system, you will need to remove the one that you have
created. The easiest way to do that is to terminate the parent process using the kill command. For
example, if the PID of the parent is 4884, you would enter “kill -9 4884”



```
F S   UID      PID    PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S 85191356 72200  72199  0  80   0 -  4602 do_wai pts/1    00:00:00 bash
0 S 85191356 74476  72200  0  80   0 -   625 do_wai pts/1    00:00:00 q5
1 Z 85191356 74477  74476  0  80   0 -     0 -      pts/1    00:00:00 q5 <defunct>
0 R 85191356 74481  74476  0  80   0 -  2522 -      pts/1    00:00:00 ps
```

Our program forks the initial process and puts the parent process into a sleep mode for 10 seconds. The child process, at the same time, tries to exit and sends an exit signal to the parent. Since the parent is on sleep mode, it can’t send the acknowledge signal back causing the child to hover in the processes list making it a zombie process. After 10 seconds, the parent process executes the ps -l command and lets the child exit.

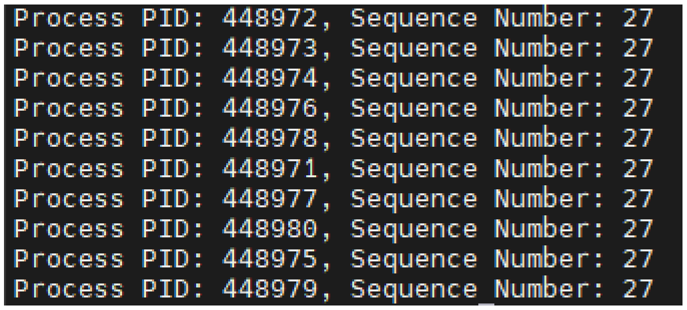
6. (20 Points) Write a simple sequence-number system through which 10 concurrent processes, P1, P2,
P3...P10 use the fork() call for process creation . Given a file, F, containing a single number, each
process must perform the following steps:

Open F
Read the sequence number N from the file
Close F
Output N and the process' PID (either on screen or test file)
Increment N by 1
Open F
Write N to F
Close F.

Describe the behavior of your program and explore the reason for this behavior. Provide evidence
for your conclusion in the form of test output. You must clearly document your code. (Partial code

provided on Canvas).
Since all of the processes race to access the sequence_number first, each of them receive the same number, and after incrementing it, they all save the same value inside of that file. Which means that the processes are not synchronized properly. Since the processes try to update the file independently, they end up competing with each other. For example, p1 tries to read the file while p2 at the same time, since this is not synchronized p2 increments based on the original position. Hence, we get the same sequence numbers for all of the PIDs.

The initial value is 27. Therefore, they all saved 28 inside of the text file.



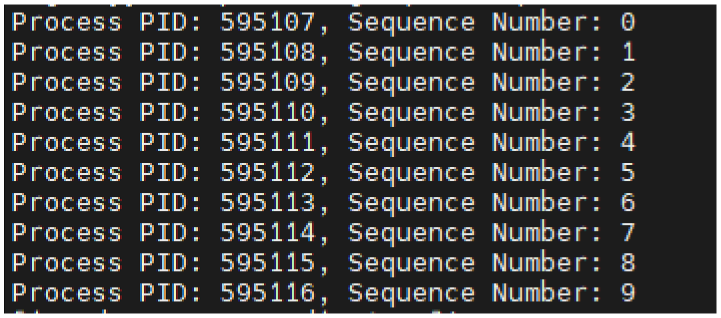
```
Process PID: 448972, Sequence Number: 27
Process PID: 448973, Sequence Number: 27
Process PID: 448974, Sequence Number: 27
Process PID: 448976, Sequence Number: 27
Process PID: 448978, Sequence Number: 27
Process PID: 448971, Sequence Number: 27
Process PID: 448977, Sequence Number: 27
Process PID: 448980, Sequence Number: 27
Process PID: 448975, Sequence Number: 27
Process PID: 448979, Sequence Number: 27
```

7. (Bonus: 10 Points) Modify the code created in the above program and fix the problem you observed.

<u>q7.c File</u>
For the bonus part, we used the POSIX IPC shared buffer and a semaphore. First, we created a shared memory buffer with read and write permissions and shared it among all 10 processes. Next, we initialized a semaphore that would lock the shared buffer, preventing other processes from accessing it while the current process increments the number value and stores it back in the buffer.

Screenshot for the IPC and semaphore approach

Initial number is 0

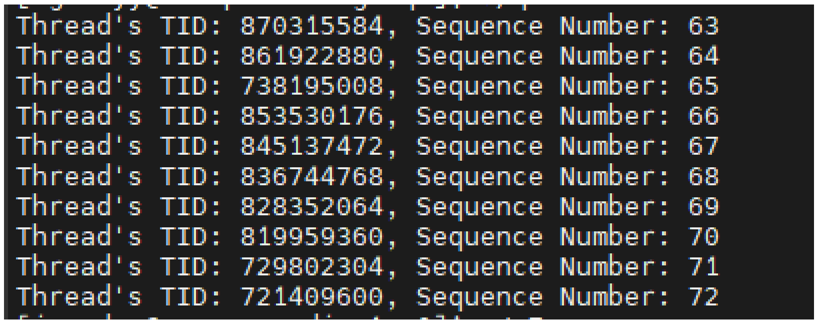


```
Process PID: 595107, Sequence Number: 0
Process PID: 595108, Sequence Number: 1
Process PID: 595109, Sequence Number: 2
Process PID: 595110, Sequence Number: 3
Process PID: 595111, Sequence Number: 4
Process PID: 595112, Sequence Number: 5
Process PID: 595113, Sequence Number: 6
Process PID: 595114, Sequence Number: 7
Process PID: 595115, Sequence Number: 8
Process PID: 595116, Sequence Number: 9
```

Next, we used multithreading programming and practiced using pthreads for synchronization, using mutex lock.

Screenshot for the threads approach

The initial value was 63.



```
Thread's TID: 870315584, Sequence Number: 63
Thread's TID: 861922880, Sequence Number: 64
Thread's TID: 738195008, Sequence Number: 65
Thread's TID: 853530176, Sequence Number: 66
Thread's TID: 845137472, Sequence Number: 67
Thread's TID: 836744768, Sequence Number: 68
Thread's TID: 828352064, Sequence Number: 69
Thread's TID: 819959360, Sequence Number: 70
Thread's TID: 729802304, Sequence Number: 71
Thread's TID: 721409600, Sequence Number: 72
```