1.Theoretical Questions

Yutian Jing

260680087

1.When we already use 2 semaphores in the producer-consumer problem why is there a need for mutex?

By using a mutex, other processes are not able to modify the global buffer when only one of the processes is in its critical section. Thus by using a mutex, the tasks become mutually exclusive and no race conditions would occur. While by using only semaphores, if there are multiple producers or consumers (multithread) that shares the same memory space, two or more processes might read or write the same memory slot at the same time because semaphores can be modified by different processes (mutex can only be unlocked by the process who locked it), resulting race conditions.

2.Is it possible that a consumer with lowest priority suffers from starvation in the 2 semaphore and 1 mutex setup for producer-consumer? Explain the situation.

It is possible. When the consuming rate is higher than the producing rate (multiple consumers and single producer), the lowest priority consumer will suffer from starvation because other consumers with higher priorities consumes up the whole full buffer, there would be nothing left for the lowest priority consumer to consume, it would stuck forever on the wait(&full), thus having a starvation situation.

3.Though binary semaphores void starvation and mutexes don't, why is it recommended to use mutex in producer-consumer to secure the critical section?

A mutex can only unlock (signal()) by the process who locked it (wait()), but a binary semaphore can be unlocked by semaphores in other processes, therefore having potential risk of multiple simultaneous critical sections causing errors, so mutex is more preferred to secure the producer-consumer critical sections.

4.Why do producer-consumer need to have 2 semaphores namely "Full" and "empty"? What complications may arise if we use only one semaphore for "Full" and rely on computed complementary value for "empty"?

A producer is also a consumer: it consumes empty slots in the buffer. By using two semaphores, two proccesses operate the buffer dependently, and the sum of empty semaphore count and full semaphore count is always equal to the buffer size. If only using full semaphore, it will be a failure if we have different producing and consuming rates. When the producing rate is higher, the producer will not stop if the buffer is full. When the consuming rate is higher, the producer will be blocked (deadlocked) if the buffer is empty since the consumer will wait on an empty full buffer.