## **PROCESS SYNCHRONIZATION**

**Chapter 6 & Chapter 7**

## The first known correct solution to the 2-process critical section problem was developed by Dekker. The solution is shown below. Prove that the solution satisfies mutual exclusion, progress and bounded waiting

## do {

## flag[i] = TRUE;

## while ( flag[j] ) {

## if (turn == j) {

## flag[i] = FALSE;

## while ( turn ==j )  do\_nothing;

## flag[i] = TRUE;

## }

## }

## CRITICAL SECTION

## turn=j;  flag[i] = FALSE;

## REMAINDER SECTION

## }

## while(true);

## 2.  Write a solution to the standard synchronization problem using TestAndSet that satisfies mutual exclusion, progress and bounded waiting. Prove that your solution indeed satisfies mutual exclusion, progress and bounded waiting.

## 3. Describe the implementation of semaphore wait() and signal() operation such that code below offers mutual exclusion to the standard synchronization problem. Show that if wait() and signal are not executed atomically then mutual exclusion may be violated. Assume that the processes use semaphore "mutex" (initialized to 1) to achieve mutual exclusion.

###### repeat

###### wait(mutex);

###### CRITICAL SECTION

###### signal(mutex);

###### REMAINDER SECTION

###### until(false);

###### 

###### Process Structure

## 4. Explain why disabling interrupts is not a good solution to the critical section problem.

## 5. Rewrite the solution given in class using semaphores to the Readers-Writers problem described here:

## The reader processors all try to read a shared variable. The writer processes write to a shared variable. There could be several readers reading the variable at the same time. However, when any one writer is writing to the shared variable, no reader can read, nor can any other writer write.

Does your solution satisfy synchronization requirement? Explain

Does your solution satisfy deadlock freedom? Explain.

Does your solution satisfy starvation freedom? Explain.

## 6. Use the semaphore solution for the Dining Philosophers problem that is given in the book to answer the following question.

## The synchronization requirement is that each chopstick can be used by one philosopher at any time. However, two philosophers that are not neighbors should be able to eat at the same time. Prove that your solution satisfies synchronization requirement and deadlock freedom. Does it satisfy starvation freedom? Discuss.

## 

## 7. Develop a semaphore solution to the Bounded Buffer problem.

## The bounded buffer problem is the one which there is producer processes and a consumer processes. The producer process produces data and places it on a buffer of size n. The consumer process consumes the data from the buffer. The producer process cannot write to the buffer, when the consumer is consuming and  when the buffer is full. The consumer process cannot consume when producer is modifying the buffer nor when the buffer is empty.

Does your solution satisfy synchronization requirement? Explain

Does your solution satisfy deadlock freedom? Explain.

Does your solution satisfy starvation freedom? Explain.

## 

## 8. Given the following semaphore solution to the Dining Philospher’s problem, explain why it does or does not satisfy deadlock freedom.

## while (true) {

## wait(chopstick[(i) mod 5];

## wait(chopstick[(i+1) mod 5];

## Eat

## signal(chopstick[(i) mod 5];

## signal(chopstick[(i+1) mod 5];

## Think

## }

## 9. Given the following solution to the Dining Philospher’s problem, explain why each of the values of mutex =1,2,3,4,5 do not work? In each case, explain which one the three requirements (synchronization requirement, deadlock freedom or starvation freedom) fails.

## while (true) {

## wait(mutex)

## wait(chopstick[(i) mod 5];

## wait(chopstick[(i+1) mod 5];

## signal(mutex);

## Eat

## signal(chopstick[(i) mod 5];

## signal(chopstick[(i+1) mod 5];

## Think

## }

## 

## 10. The Sleeping Barber Problem: A barber shop consists of a waiting room with n chairs and a barber room with one barber chair. It there are no customers, to be served, the barber goes to sleep. If a customer enters the barbershop and all chairs are occupied, then the customer leaves the shop. If the barber is busy, but chairs are available, then the customer sits in one of the free chairs.  If the barber is asleep, the customer wakes up the barber.

## Write a solution to the problem using semaphores only.

Does your solution satisfy synchronization requirement? Explain

Does your solution satisfy deadlock freedom? Explain.

Does your solution satisfy starvation freedom? Explain.

## 11. The Sleeping Barber Problem: A barber shop consists of a waiting room with n chairs and a barber room with one barber chair. It there are no customers, to be served, the barber goes to sleep. If a customer enters the barbershop and all chairs are occupied, then the customer leaves the shop. If the barber is busy, but chairs are available, then the customer sits in one of the free chairs.  If the barber is asleep, the customer wakes up the barber.

## Write a solution to the problem using semaphores.

Does your solution satisfy synchronization requirement? Explain

Does your solution satisfy deadlock freedom? Explain.

Does your solution satisfy starvation freedom? Explain.

## 13. The Cigarette Smokers’ Problem: Consider a system with three smoker processes and one agent process. Each smoker continuously rolls a cigarette and then smokes it. But to roll and smoke a cigarette, the smoker needs three ingredients: tobacco, paper and matches. One of the smoker processes has paper, another has tobacco and third has matches. The agent has an infinite supply of all three materials. The agent places two of the materials on the table. The smoker that has the third ingredient makes and smokes a cigarette and signals the agent upon completion.  The agent then puts out another two ingredients and the process repeats.

## Write a solution to the problem using semaphores only.

Does your solution satisfy synchronization requirement? Explain

Does your solution satisfy deadlock freedom? Explain.

Does your solution satisfy starvation freedom? Explain.