

BIRZEIT UNIVERSITY

Electrical and Computer Engineering Department

ENCS339 Operating Systems

# Homework 2 Solution

## Tareq Shannak – 1181404

# Deadlock Prevention

1. Describe a real-life deadlock situation NOT in traffic management. Explain why it satisfies the four necessary conditions (mutual exclusion, hold-and-wait, non-preemption, circular wait). How does one resolve that situation? Which condition becomes false?

Without getting the experience, you cannot get the work and you cannot get the experience without having a job.

**Mutual exclusion**: The head of the company prefers only experienced people, but the people will get experience only if they provided the job and it results in the occurrence of the deadlock.

**Hold and wait**: Both of them (head and the candidate) fail to withdraw and keep their ground, so it is holding.

**No Preemption**: For solving the deadlock problem, the head can remove the experience category but he/she will not do that and again prefer experienced candidates.

**Circular wait**: Both of them refuses to withdraw and wait for each other to withdraw, so that they can complete their task. This is circular wait.

Here one can resolve the situation by eliminating any one of the above condition. For example, if the head prefers zero experience candidates for certain jobs, freshers can apply to it and get selected. Therefore they gain experience and can move to higher positions, thus avoiding mutual exclusion condition in deadlock.

1. Give a real life example (better related to the example you gave for a) of a **starvation** situation and show how it differs from the deadlock in part a.

If I can’t has job as a freshman because I don’t have experience and I don’t have experience because I didn’t have a job, then I will never have a job and experience.

1. Give an example, where the system is not in a safe state, but if the processes of the system are allowed to be executed, then they will be successfully complete (unsafe state without Deadlock).

If the available resources A B = (2, 2)

|  |  |  |  |
| --- | --- | --- | --- |
|  | Allocation A B | Max A B | Need A B |
| P0 | 1 2 | 4 5 | 3 3 |
| P1 | 2 2 | 4 6 | 2 4 |

This is an unsafe system that may lead to a deadlock, put if P0 gave up the B resources it has after sometime the need in P0 will become (3, 5), the available resources A and B will become (2, 4) which is enough to get P1 executed. After P1 is done, the available resources A and B will become (4, 6) which is enough to get P0 executed and there will be no deadlock in this unsafe system.

#  Deadlock Avoidance

Consider the following snapshot of a system:



1. Is the system in a safe or unsafe state? Execute the Banker's algorithm to check. Show the different values of the work vector after every iteration. What is the sequence of processes that the algorithm implicitly created? Any other ordering possible in this case?

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Allocation A B C | Max A B C | Need A B C | Initial Available = 2 0 0 |
| P0 | 1 4 1 | 5 6 2 | 4 2 1 | **3 :** Available = 5 6 2 |
| P1 | 0 1 0 | 2 1 0 | 2 0 0 | **1 :** Available = 2 1 0 |
| P2 | 1 1 1 | 5 6 1 | 4 5 0 | **4 :** Available = 6 7 3 |
| P3 | 3 2 3 | 9 9 6 | 6 7 3 | **5 :** Available = 9 9 6 |
| P4 | 2 1 1 | 3 2 1 | 1 1 0 | **2 :** Available = 4 2 1 |

 The system is in safe state, the sequence P1 -> P4 -> P0 -> P2 -> P3. There is no another possible order.

1. Do the same if P0 had the allocation vector (3 2 2) instead of (1 4 0). Which processes are NOT able to finish (if any)?

 P0, P2, P3 aren’t able to finish.

1. If a request from process P0 [in the original case] arrives for (1, 0, 0,), can the request be granted immediately? Justify your answer.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Allocation A B C | Max A B C | Need A B C | Initial Available = 1 0 0 |
| P0 | 2 4 1 | 5 6 2 | 3 2 1 | **No process can be work with this available resource** |
| P1 | 0 1 0 | 2 1 0 | 2 0 0 |
| P2 | 1 1 1 | 5 6 1 | 4 5 0 |
| P3 | 3 2 3 | 9 9 6 | 6 7 3 |
| P4 | 2 1 1 | 3 2 1 | 1 1 0 |

P0 can’t request (1, 0, 0).

1. If instead a request from process P4 arrives for (2, 0, 0,), can the request be granted immediately? Justify your answer.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Allocation A B C | Max A B C | Need A B C | Initial Available = 0 0 0 |
| P0 | 1 4 1 | 5 6 2 | 4 2 1 | **Need[i] can’t be in negative****Initial available can’t make any process work** |
| P1 | 0 1 0 | 2 1 0 | 2 0 0 |
| P2 | 1 1 1 | 5 6 1 | 4 5 0 |
| P3 | 3 2 3 | 9 9 6 | 6 7 3 |
| P4 | 4 1 1 | 3 2 1 | -1 1 0 |

P4 can’t request (2, 0, 0).

1. Can we reduce the overall number of resource instances and still be in a safe state? Why? Why Not?

No we can’t reduce the resources because the overall number is (9, 9, 6) and it equals to the maximum number of P3, so if we reduce the resources, P3 can’t be executed.

1. Can we increase the number of resource instances so that we are always in safe states without the need for testing? Why? Why Not?

Yes of course, if the available resources are more than the maximum need of all processes, then it is always in safe state.

Synchronization

1. Is the following a correct way for solving the critical section problem (modified Peterson). Why and why not?

Process Pi:

do {

flag[i] = TRUE;

turn = i;

while (flag[j] && turn == j);

<critical section>

flag[i] = FALSE;

<remainder section>

 } while (TRUE);

1. Three processes are involved in printing a file (pictured below). Process A reads the file data from the disk to Buffer 1, Process B copies the data from Buffer 1 to Buffer 2, finally Process C takes the data from Buffer 2 and print it.



Assume all three processes operate on one (file) record at a time. Buffer capacity is one record. Show that the program below coordinates the work of the three processes based on the 4 semaphores.

Answer:

semaphore empty1 = 1;

semaphore empty2 = 1;

semaphore full11 = 0;

semaphore full12 = 0;

Process\_A () {

 while(1) {

 wait(empty1);

 read(next\_file(), Buffer\_1);

 signal(full1);

}

 }

Process\_B () {

while(1) {

wait(full1);

wait(empty2);

copy(Buffer\_2, Buffer\_1);

signal(empty1);

signal(full2);

}

}

Process\_C () {

while(1) {

wait(full2);

print(Buffer\_2);

signal(empty2);

}

 }

At first the program will enter procces A , first it will check the empty1 , it will find that its equal to 1 meaning that buffer\_1 is empty and ready to get the data, since the data is now stored in in the buffer\_1 , the full1 semaphore will change to 1 meaning the buffer is full of data and ready to be copied.

After that, the program will enter process B , which will check if buffer\_1 is full of data , since its 1 it will keep going and check if buffer\_2 is empty , if it is the program will copy the data from buffer\_1 and store it in buffer\_2 and make empty1 semaphore 1 meaning its empty, and full2 semaphore is 1 meaining its full.

At last the program will enter process 3 , check if buffer\_2 has data, since semaphore full2 is 1 so it will print the data and make semaphore empty2 1 meaning its empty.

If the program enters process B before A the semaphore full1 will be 0 so it cant continue, if it enters process C before A or B , the semaphore full2 will be 0 so it cant continue.

So these semaphores makes sure that the only correct order is A then B then C, nothing else will work.