COMP 3500: Homework 2

Points Possible: 100
Note: You do not need to submit hard copies.

There should be no collaboration among students. A student shouldn't share any project code with any other student. Collaborations among students in any form will be treated as a serious violation of the University's academic integrity code.

Goals:

- To understand the principles of deadlocks.
- To learn how to solve deadlock and starvation problems.
- To collaborate and discuss deadlock problems with your group members.

Questions:

1. [40 points]z

In the code below, three processes are competing for six resources labeled A to F.

a. Using a resource allocation graph (Figures 6.5 and 6.6), show the possibility of a deadlock in this implementation.

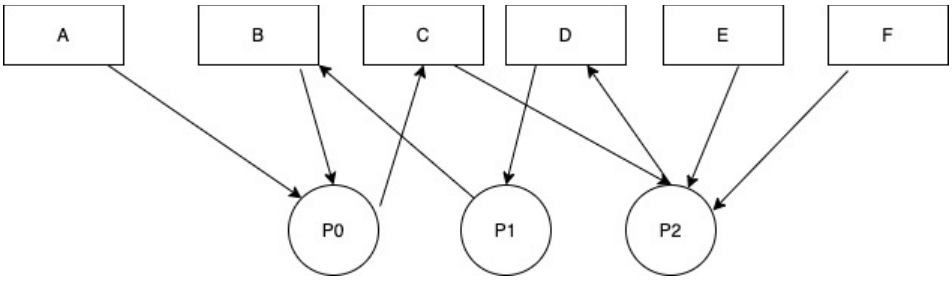
b. Modify the order of some of the get requests to prevent the possibility of any deadlock. You cannot move requests across procedures, only change the order inside each procedure. Use a resource allocation graph to justify your answer.

```
void P0()
{
  while (true) {
    get(A);
    get(B);
    get(C);
    // critical region:
    // use A, B, C
    release(A);
    release(B);
    release(C);
  }
}
```

```
void P1()
{
  while (true) {
    get(D);
    get(E);
    get(B);
    // critical region:
    // use D, E, B
    release(D);
    release(E);
    release(B);
  }
}
```

```
void P2()
{
  while (true) {
    get(C);
    get(F);
    get(D);
    // critical region:
    // use C, F, D
    release(C);
    release(F);
    release(D);
  }
}
```

A. The order I chose was p0 -get(a), p1-get(d), p2-get(c), p0-get(b), p1-get(e), p2-get(f), p0-get(c), p1-get(b), p2-get(d)

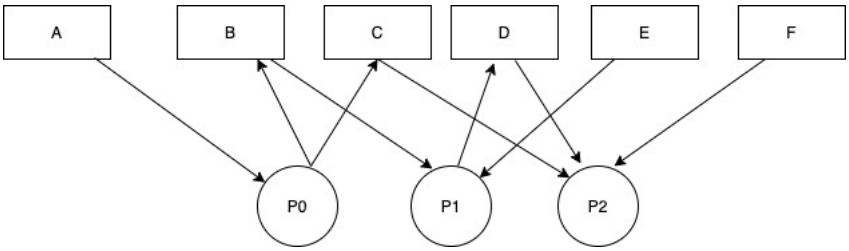


Since there is a cycle in the resource allocation graph, and there's only one instance per resource, this indicates the presence of a deadlock.

b. We need to make sure each process tries to use the resources in the same order.

p0:	p1:	p2:
get(a)	get(b)	get(c)
get(b)	get(d)	get(d)
get(c)	get(e)	get(f)
release(a)	release(b)	release(c)
release(b)	release(d)	release(d)
release(c)	release(e)	release(f)

This ensures that if a process attempts to access a resource that Is unavailable, it will be forced to wait until the process holding it releases it before continuing. Now it is guaranteed there will be no deadlock, and no resource allocation graph will contain a cycle. i.e



2. [20 points]
Suppose the following two processes, `foo` and `bar` are executed concurrently and share the semaphore variables `S` and `R` (each initialized to 1) and the integer variable `x` (initialized to 0)

```
void foo( ) {
   do {
     semWait(S);
     semWait(R);
     x++;
     semSignal(S);
     SemSignal(R);
   } while (1);
}
```

```
void bar( ) {
do {
    semWait(R);
    semWait(S);
    x--;
    semSignal(S;
    SemSignal(R);
} while (1);
}
```

Can the concurrent execution of these two processes result in one or both being blocked forever? If your answer is yes, please give an execution sequence in which one or both are blocked forever.

Yes, both processes can be blocked, if they run in parallel.

If foo calls semWait(s), the value of S will now be 0.

then bar calls semWait(R) is now 0.

now when each process tries to call semwait again, they will be stuck indefinitely waiting on a signal.

However, Foo and bar cannot be blocked individually, they will always be blocked together if they are blocked, because of the alternate calls on semwait on the respected semaphores.

3. [20 points]
What is the difference among deadlock avoidance, detection, and prevention?
Deadlock prevention is a static policy, that eliminates on of the conditions for deadlock before the program begins. Deadlock avoidance, makes the appropriate dynamic choices based on the current state of resource allocation. Deadlock detection attempts to detect the presence of a dead lock, and then take action to recover.

4. [20 points]
Consider a system consisting of four resources of the same type that are shared by three processes, each of which needs at most two resources. Show that the system is deadlock-free.

If the system is deadlocked, that means each process is holding a resource, and is waiting on another. However, since there are 3 processes, and 4 resources, one process will be able to hold two no matter what. Then it can return its resources for the other processes to use. Therefore, there will never be a deadlock.

Submission:

- A heading at the top of your file contains your name and your Auburn UserIDs.
- Submit your solution as a single PDF file named as “hw2.pdf” through Canvas
- File formats other than PDF will not be accepted by Canvas.

Late Submission Penalty:

- Ten percent (10%) penalty per day for late submission. For example, an assignment submitted after the deadline but up to 1 day (24 hours) late can achieve a maximum of 90% of points allocated for the assignment. An assignment submitted after the deadline but up to 2 days (48 hours) late can achieve a maximum of 80% of points allocated for the assignment.
- Assignment submitted more than 3 days (72 hours) after the deadline will not be graded.

Rebuttal period:

- You will be given a period of one week (i.e., 7 days) to read and respond to the comments and grades of your homework or project assignment. The TA may use this opportunity to address any concern and question you have. The TA also may ask for additional information from you regarding your homework or project.