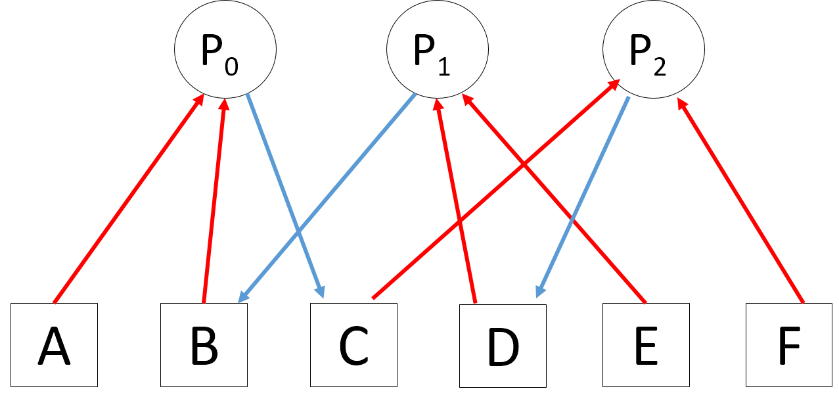
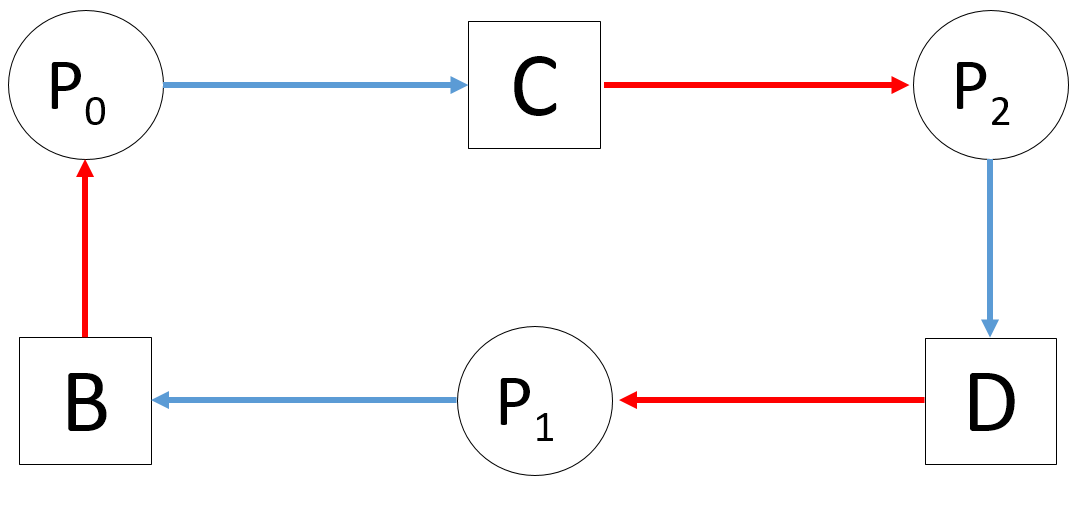
Bethany Edgar

COMP 3500

Homework 2

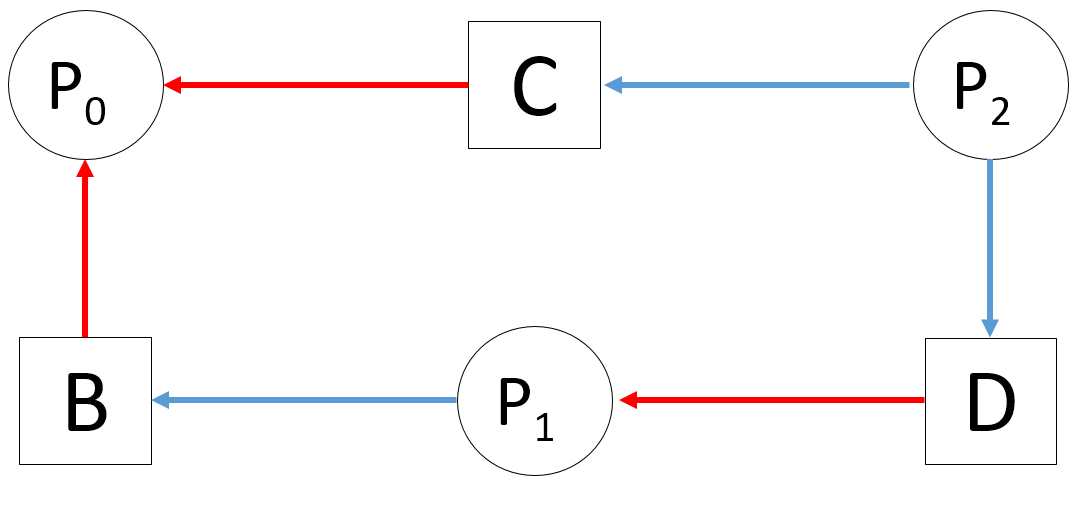


1. Assuming single instance resources, no preemption, and mutual exclusion (locks), this process exhibits hold and wait as well as circular wait. See simplified resource allocation graph featuring only the resources involved in deadlock below. Deadlock can occur.



|  |  |  |
| --- | --- | --- |
| void P0()  {  while (true) {  get(A);  get(B);  get(C);  //critical region:  // use A, B, C  release(C);  release(B);  release(A);  signal(mutex);  signal(mutex);  }  } | void P1()  {  while (true) {  get(E);  wait(mutex);  get(D);  get(B);  //critical region:  // use E, D, B  release(B);  release(D);  release(E);  }  } | void P2()  {  while (true) {  get(F);  wait(mutex);  get(C);  get(D);  //critical region:  // use F, C, D  release(D);  release(C);  release(F);  }  } |

This imposes an ordering on the processes where P0 always goes first, then the circular wait is broken and P1 and P2 can execute concurrently since they both get unlocked by P0.

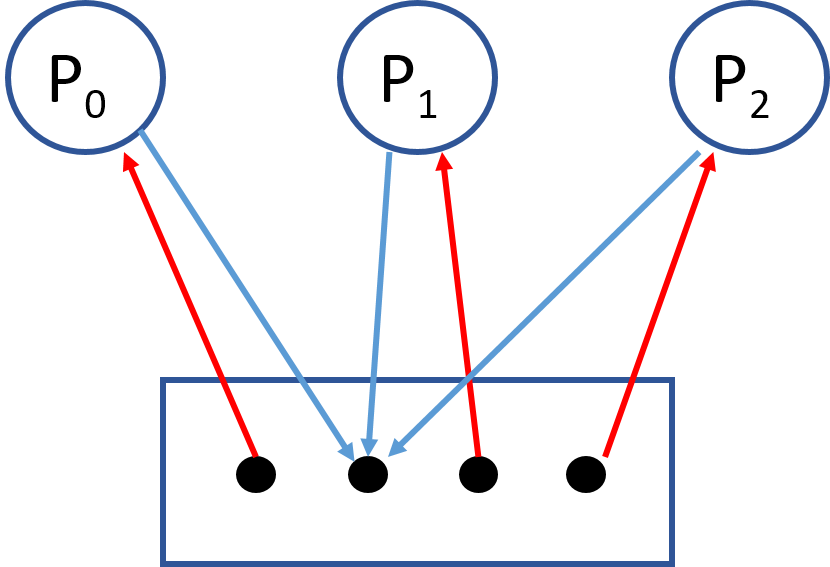


1. Yes, if the first line of foo executes and consumes the initial wait signal for S, then switches to the bar process and the initial signal to R is consumed. Now both R and S are waiting for signals that can never get sent as they cannot proceed to the later portion of the process.
2. Deadlock Avoidance: The avoidance technique deals with deadlocks dynamically, during the course of execution and is based on the current state of the process. Using this technique the system should never enter deadlock.

Deadlock Detection: Detection waits for a deadlock to occur, and then it breaks the deadlock by removing one of the four necessary conditions to create deadlock.

Deadlock Prevention: Deals with the code design process. It is coding to specifically eliminate any possibility of deadlock. This is static deadlock prevention.

1. If all hold one instance of the resource, and wait for another, then there will be one instance leftover (4-3=1). Then whichever process grabs the last instance first will complete and thus free up two instances. So then the last two processes can run concurrently. This could also be done in the opposite order, with two processes completing first then the remaining process finishing last.



This illustrates that there is no circular wait as the instances of the single resource can be used interchangeably and there are more instances than there are processes.