Christian KHOURY

Interprocess Synchronization

Concurrent Access To Shared Memory : Race Problems

If a memory variable is shared by different processes and these processes modify it concurrently, then this might lead to a final erroneous result ! The goal in the following exercise is to show these possible errors.

1. Using two tasks, create a shared variable 'i' and initialize it 65; one task should increment the variable and the other one should decrement it
2. Explain why the following code could lead to an error. Reg = i

   sleep(for_some_time) // your choice Reg++ (or Reg--); // depending on

   the task i = Reg;

Solving the Problem : Synchronizing access using semaphores

1. Use semaphores to enforce mutual exclusion and solve the race problem in the first exercise (sem_init (sem_open for macOS users), sem_wait, sem_post)

   a. What if we had more than two processes ? Is there something else to do to enforce mutual exclusion ? Explain and experiment using three processes.

2. A deadlock is a situation in which a process is waiting for some resource held by another process waiting for it to release another resource, thereby forming a loop of blocked processes ! Use semaphores to force a deadlock situation using three processes.

3. Use semaphores to run 3 different applications (firefox, emacs, vi) in a predefined sequence no matter in which order they are launched.

4. Use sempahores to implement the following parallelized calculation (a+b)*(c-d)*(e+f)
   T1 runs (a+b) and stores the result in a shared table (1st available spot)
   T2 runs (c+d) and stores the result in a shared table (1st available spot)
   T3 runs (e+f) and stores the result in a shared table (1st available spot)
   T4 waits for two tasks to end and does the corresponding calculation
   T4 waits for the remaining task to end and does the final calculation then displays the result