CSE231 - Operating Systems

Assignment-4

Name: Ananya Lohani
Roll Number: 2019018
Branch: CSE

Description:

I have implemented my own counting semaphore by the struct my_semaphore. I have implemented two versions of wait() and signal(): one which blocks the thread if the resource isn't available(blocking) and one which directly returns without blocking the thread(non-blocking).

Blocking Wait:

The blocking wait version makes use of a conditional variable within my_semaphore. The philosophers try to acquire the left fork, right fork and the sauce bowls in that order. We first make the acquirement of forks atomic by using another semaphore called mutex and locking it. In the wait() function, if the value of the semaphore is 0, i.e. the resource isn't available, the thread blocks through the method pthread_cond_wait() until the resource becomes available. Else, it decrements the semaphore value and returns. In the signal() function, if the value of semaphore is less than the max_value(=1 in our case), then the value is incremented. If the value is greater than zero, we call pthread_cond_signal() which informs the conditional variable that a resource has become available. After acquiring both the forks atomically, we signal mutex so that the acquirement of bowls is not atomic. Then we acquire the bowls. After printing the value of the forks acquired by a philosopher, we signal all the resources.

Non-blocking Wait:

The non-blocking wait differs from the blocking wait in only the implementation of the wait() and signal() functions. In the wait() function, I call the pthread_mutex_trylock() function which tests if the lock is available to be acquired. If not, I return 1. If it's available then check if the semaphore value is greater than zero. If it's not, return 1. If it is, then decrement the semaphore value and return 0. The signal() function checks if the value of the semaphore is less than the max_value and increments it if it's not.