NAME

twoPhaseLock - Dining philosopher's problem with two-phase locking

SYNOPSIS

This is an implementation of two phase locking algorithm to solve the concurrency issues in the dining philosophers problem in C. pthreads are used to simulate the philosophers. Semaphors are used to lock the chopsticks and avoid concurrency issues.

Deadlock prevention algorithm (pseudo code):

acquire mutex lock
check whether left and right chopsticks are free
pick chopsticks up/acquire their semaphores/wait
release mutex lock

RUN

./twoPhaseLock *<number_of_philosophers>* *<type_of_print>*

OPTIONS

<number_of_philosophers>

Integer input to define number of philosophers considered.

<type_of_print>

Integer input to define print style. Input range {0,1,2}.

0 - No print
1 - Print verbose
2 - Print only current state of all philosophers

Overall

./twoPhaseLock 6 2

COMPILATION

gcc twoPhaseLock.c -o twoPhaseLock -lpthread

FILES

/twoPhaseLock

Executable file for the dining philosopher problem.

/twoPhaseLock.c

Source file

METHODS

void *philosopher(*void *num*);

Main philosopher method contains
an infinite loop inside which
random sleep times
pick up chopsticks with function, pickupSticks()
random sleep times
drop chopsticks with function, dropSticks()

void pickupSticks(*int x*);

pickupSticks()
acquire mutex lock
check whether left and right chopsticks are free
pick chopsticks up/acquire their semaphores/wait
release mutex lock

void dropSticks(*int x*);

dropSticks()
acquire mutex lock
release chopsticks/signal/post
release mutex lock

void printAll(*int philosopherIndex, int currentStatus*);

printAll()
Depending on input 1 or 0
Print verbose, selective, nothing

void runPhilosopherModule();

runPhilosopherModule
Driver module for philosophers

Status initialisations

void initialiseStatus();
void setMode();
void setPhilosopherCount(*int size*);
void initialiseSemaphore();

AUTHORS

Atish Majumdar
Prasanna Natarajan
Vedant Chakravarthy