EECS338 - Operating Systems - HW#3

Mark Schultz - mxs802

February 19, 2011

1

a) By emiminating the starred line, processes no longer wait for other processes to finish choosing. If we don't wait for all processes to finish choosing, more than one process may enter the CS. For example, P_i chooses a number and enters the CS. becuase it has entered the CS but hasn't waited for the other processes to choose, it is possible another process P_k will choose a lower number therefore passing all other checks and entering the critical section.

b) This algorithm is FIFO because of the number check. Before entering the critical section, each process checks if there are any processes with a smaller number and lets all of them execute. Because of this check, a process cannot jump ahead in line, every process must execute in order of their numbers. Therefore, the bakery algorithm must be FIFO.

c) The bakery algorithm should satisfy bounded waiting because when it chooses a number that determins its place in line to execute. After a process picks its number, no other process should be able to cut in line so the bounded waiting value should be P_i's place in line times the time it takes to execute the critical section.

2

semaphore add = 0
semaphore divide = 0
parbegin
begin: $T1 = A * B$
signal(add)
begin: $T2 = C * D$
signal(add)
begin: $T3 = E * F$
signal(divide)
begin: wait(add)
wait(add)
$T4 = T1 + T2$
signal(divide)
begin: wait(divide)

wait(divide)
$out = T4/T3$
parend

3

The dinig philosophers cannot enter a livelock. Each philosopher, or process, will always either put the chopsticks down or wait to pick them both up. A livelock would only occur if the philosophers were able to pick up one chopstick then put it back down.

4

Notes: This solution does not satisfy bounded waiting in several spots.

//Global vars
var train, car: semaphores
var xing: int
var txing: int

P_i**: repeat**
bakers algorithm instead of dropping into the CS it calls cross(i); /* the result of this is that only one car wi
until False

$cross_i$**:**
if $xing = 4$ **then** $wait(car)$ **//if 4 crossing, wait for one to exit**
$xing++$ // if xing is less then 4, no need to wait
$number[i] := 0$ // release next car to enter cross(i)
if $light \neq green$ **then** $wait(train)$ **// there is a train crossing or going to cross soon**
... CROSS ...
$xing--$
$signal(car)$
end

$train_i$**:**
$light = yellow$
$txing++$
$light = red$
... PASS ... // from the description of the problem, trains can never collide so need no queueing etc.
$txing--$
while $txing \neq 0$ **do nop // wait for all trains to pass**
$light = green$
$signal(train)$
end