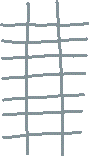
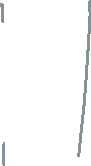
Ýmir Þórleifsson

HW4

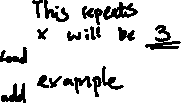
# 4.1.2

a) We would expect the outcome to be 6 if each prosess is executed after each other with buffer.

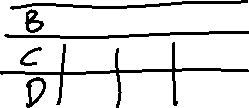
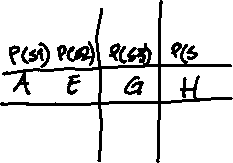


b) As soon as the processes run interleaved, the outcome will be less than the expected outcome.

c)



# 4.2.1



# 4.2.3

a) Indefinitely if f1() and f2() are running concurrently because after each cycle the values of the variables reset, so the values remain s1 = s2 = d = 1 and c1 = c2 = 0

b) 0, because in line 3 of f2(), P(d) is blocked until f1() line 8 of V(d)

c) no, because they use sophomores that prevent starvation

# 4.3.1

a)

Pb(sb): R = 0

Do SWAP(R, sb) while (R == 0)

Vb(sb): sb = 1

b)

Pb(sb): L: TSB(sb, L)

Vb(sb): sb = 1

# 4.4.1

x = 10 y = 2

m.A()

1) x 🡪 11

2) ignore signal

3) y 🡪 9

x = 11 y = 9

m.B()

4) condition true

5) x 🡪 10

x = 10 y = 9

m.B()

4) condition false

6) placed in wait que

x = 10 y = 9 wait que = m.B()

m.B()

4) condition false

6) placed in wait que

x = 10 y = 9 wait que = m.B(), m.B()

m.B()

4) condition false

6) placed in wait que

x = 10 y = 9 wait que = m.B(), m.B(), m.B()

m.A()

1) x 🡪 11

2) execute signal m.B()

7) x 🡪10

x = 10 y = 9 wait que = m.B(), m.B()

m.A()

1) x 🡪 11

2) execute signal m.B()

7) x 🡪10

x = 10 y = 9 wait que = m.B()

x changes from 10 to 10, y changes from 2 to 9

# 4.4.4

f(x) {

if (x) P(c1)

x = x + 1

V(c2)

x = 0

}

# 4.4.6

a) if (x < 0) c.wait

exit(0)

b) x = 1

c.notify

c.notify

x = h(x)

# 4.5.1

a) w1, w2, r1 & r2, w3, w4. r1 and r2 will read concurrently.

b) They both go to a wait list and can read along with r1 and r2 when w2 is finished writing.

# 4.5.3

Yes it avoids corruption successfully. Starvation is possible if too many read threads arrive then there is no chance for write to execute.

# 4.5.6

Let’s check philosopher 0:

P(mutate) success

P(f[i]) success

P(f[i + 1 % 5] success

V(mutate) success

eat

Philosopher 0 is eating alright

Philosopher 1:

P(mutate) success

P(f[i]) unsuccess

BLOCKED

Unable to eat because philosopher 0 is using fork

Philosopher 2:

P(mutate) unsuccess

BLOCKED

Unable to eat because philosopher 1 is thinking

Thus, only one philosopher can eat at a time and does not satisfy all requirements of the dining philosophers’ problem.

# 4.5.8

a) Now it will not stop if you want to get off on same floor as you are on while going up.

b) Now any request of stopping on same floor as you are on while going down is canceled.