Assignment 1

# CS 374 – Operating Systems

## DUE: Monday, October 4

**Please type up answers to the following problems. An upload link will be provided on Canvas.**

1. Problem 1 in Bic & Shaw, page 66.

p4

p3

p6

p1

p5

p2

p7

p1

p2

p3

p4

p5

1. Problem 2 in Bic & Shaw, page 67.

cobegin

W = X1 \* X2;

//

V = X3 \* X4;

cobegin

Y = V \* X5;

//

Z = V \* X6;

coend;

coend;

cobegin

Y = W \* Y;

//

Z = W \* Z;

coend;

A = Y + Z;

1. Problem 3 in Bic & Shaw, page 67.

a)

cobegin

cobegin

p3;

//

p1;

cobegin

p5;

//

p6;

coend;

coend;

p2;

//

p4;

coend;

b)

t1 = 3; t2 = 2;

fork L1; fork L3; fork L4; quit;

L1: p1; fork L5; fork L6; quit;

L2: p2; join t2, FIN; quit;

L3: p3; join t1, L2; quit;

L4: p4; join t2, FIN; quit;

L5: p5; join t1, L2; quit;

L6: p6; join t1, L2; quit;

FIN: …

1. Problem 6 in Bic & Shaw, page 67.

a) forall ( i:1..n ) {

for ( j=1; j<=m; j++) {

A[i][j] = 0;

for ( k=1; k<=r; k++ )

A[i][j] = A[i][j] + B[i][k]\*C[k][j] ;

}

}

b) exchange i and j in the code above

1. Consider the following precedence graph:

p5

p1

p8

p2

p3

p7

p4

p9

p6

* 1. Express this graph using *fork*, *join*, and *quit* primitives

t1 = 2; t2 = 2;

fork L1; fork L2; fork L3; quit

L1: p1; fork L4; quit;

L2: p2; fork L6; quit;

L3: p3; fork L8; quit;

L4: p4; fork L5; quit;

L5: p5; join t1, L7; quit;

L6: p6; join t1, L7; quit;

L7: p7; join t2, FIN; quit;

L8: p8; fork L9; quit;

L9: p9; join t2, FIN; quit;

FIN: …

* 1. Express this graph using a **single** cobegin-coend block with semaphores to control the precedence

semaphore S1 = 0; S2 = 0; S3 = 0; S4 = 0; S6 = 0; S8 = 0;

cobegin

p1; V(S1);

//

p2; V(S2);

//

p3; V(S3);

//

P(S1); p4; V(S4);

//

P(S4); p5; V(S6);

//

P(S2); p6; V(S6);

//

P(S6); P(S6); p7;

//

P(S3); p8; V(S8);

//

P(S8); p9;

coend;

1. Refer to the code segment below. It might be helpful to think of the expressions as comprising large matrix operations. Note that operations are frequently dependent on the completion of previous operations: for example, Q1 cannot be calculated until M2 has been calculated.
   1. Express the code as a process flow graph maintaining the expressed precedence of operations (eg: M1 must be calculated before QR2 is calculated). Use the **left hand sides** of the equation to label processes in your process flow graph. NOTE: part a) is a bit tricky—you will need to use some empty (or *epsilon* *transition*) arcs—that is, arcs not labeled by processes—to get the best graph. You might have to draw the graph more than once to reduce crossed arcs (you won’t eliminate them).

ε

M1

QR3

M2

ε

Q1

QR2

QR1

Z1

* 1. Implement the process flow graph using fork, join, and quit. Ensure that the maximum parallelism is achieved in both parts of this problem! If the graph from the first part is correct, this part is actually easy.

**M1** = A1 \* A2;

**M2** = (A1+A2)\*B1;

**QR2** = M1\*A1;

**Q1** = M2 + B2;

**QR1** = M2-M1;

**QR3** = A1\*B1;

**Z1**=QR3-QR1;

t1 = 2; t2 = 2; t3 = 3;

fork LQR3; fork LM1; fork LM2; quit;

LQR3: QR3 = A1 \* B1; join t2, LZ1; quit;

LM1: M1 = A1 \* A2; fork LQR2; join t1, LQR1; quit;

LM2: M2 = (A1 + A2) \* B1; fork LQ1; join t1, LQR1; quit;

LQR1: QR1 = M1 – M2; join t2, LZ1; quit;

LQR2: QR2 = M1 \* A1; join t3, FIN; quit;

LQ1: Q1 = M2 + B2; join t3, FIN; quit;

LZ1: Z1 = QR3 – QR1; join t3, FIN; quit;

FIN: …