Cal Poly Humboldt

Department of Computer Science

CS 374: Operating Systems

**Assignment 2**

Please upload solutions to the homework on Canvas. You can type answers directly into this document in the appropriate places. **DUE: October 11.**

1. Consider the Hoare monitor code below. Assume that 4 processes call the methods in the following order: hdl.Huey(), hdl.Louie(), hdl.Dewey(), hdl.Louie(). If multiple processes restart, execute Huey before Dewey before Louie. Please trace the execution of the code, and **note the values of the variables for each line of code executed** in the table given.

**monitor** hdl {

// The Huey, Dewey, and Louie monitor!

**int** money = 0, food = 0, mischief = 10;

condition trouble, mess, anger\_donald;

Huey( ) {

(1) money = money + mischief/5;

(2) if (mischief > 6) trouble.wait();

(3) mischief = mischief – 2; mess.wait();

(4) food++;

}

Dewey( ) {

(5) food = food + 2;

(6) if (mischief > 0) anger\_donald.signal();

(7) trouble.signal();

(8) mischief = mischief – 1;

}

Louie( ) {

(9) if (money > food) {

(10) food--; mischief++; anger\_donald.wait();}

(11) else mischief = mischief – 2;

(12) mess.signal();

(13) food = food + 3;

} }

Condition Variable Queues (optional, to help tracing):

**Trouble Mess Anger\_Donald**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Line #:** | init |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **food:** | 0 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **money:** | 0 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **mischief:** | 10 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Some additional questions about hdl:

1. We did not go back to the other process once we were past the trouble.wait()in hdl.Huey(). Why not?
2. Why are none of the condition variables initialized to values like the integers are?
3. If the exit of a process might awaken multiple other processes, what determines their order of execution?
4. In this problem, all processes completed successfully. Is it possible that at the end of a series of calls, that some processes may not have completed? (HINT: where might they be if they didn’t complete?)
5. Recall that each of the five philosophers, j, in the Dining Philosophers problem execute the following code segment:

P(j) {

P(fork[j]);

P(fork[(j + 1) % 5];

eat;

V(fork[j]);

V(fork[(j + 1) % 5];

think( );

}

Suppose that instead of a particular fork being shared between a pair of philosophers, the five forks are placed in the middle of the table for all to share. Rewrite the code to take into account the forks are not distinct any more, and make sure you implement a simple protection mechanism to prevent deadlock (ignore starvation issues). You are free to use an if( P( ) ) test on a semaphore to do this.

1. The following expression describes the serial/parallel precedence relationship among several processes:

S( P( S( p1, p2 ), p3 ), p4, P( p5, S( p6, P( p7, p8 )) ) )

Transform this expression into a program using:

1. *Cobegin/coend* (or explain why this is not possible)
2. *Fork, join* and *quit* primitives
3. Below is a program expressed using *fork*, *join*, and *quit*.
   1. Draw the *process flow graph* for the program.
   2. Either express the program using *S/P notation*, or explain clearly why this is not possible.

(10 points)

T1 = 2; T2 = 3;

fork L1; fork L2; fork L3; quit;

L1: p1; fork L5; fork L4; quit;

L2: p2; join t1, L6; quit;

L3: p3; join T2, FIN; quit;

L4: p4; join t1, L6; quit;

L5: p5; join T2, FIN; quit;

L6: p6; join T2, FIN; quit;

FIN: …

1. Consider the following process flow graph:

p6

p2

p4

p1

p3

p5

1. Express the graph as a single cobegin/coend block with all process coordination accomplished with semaphores.
2. Express the graph using nested cobegin/coend blocks (no semaphores).
3. Consider the following rendezvous code:

while (1) {

select {

when a==TRUE :

accept A()

{ function1(); b = FALSE; }

when b==TRUE :

accept B()

{ function2(); a = FALSE; }

else { a = TRUE; b = TRUE; }

Assume that there aren’t outstanding calls to A or B when the code is first executed. Thereafter, the following calls arrive in the given order [in less time than it takes to do a pass through the code]: A( ), B( ), B( ), A( ), A( ), B( ).

1. In which order will the calls be accepted?
2. Can a caller of A (or of B) be starved? If so, explain how.