CIS450/ECE478, Solutions for homework #1

1. 15 new processes will be created.

The original process P0 starts with i = 0, loops 4 times and creates 4 child processes, P1 with i = 1, P2 with i = 2, P3 with i = 3, and P4 with i=4.

P1 starts with i = 1, continues the loop 3 times and creates 3 child processes, P5 with i = 2, P6 with i = 3, and P7 with i=4.

P5 starts with i = 2, continues the loop 2 times and creates 2 child processes, P8 with i = 3, P9 with i = 4.

P6 starts with i = 3, continues the loop one time and creates 1 child process, P10 with i = 4.

P8 starts with i = 3, continues the loop one time and creates 1 child process, P11 with i = 4.

P2 starts with i = 2, continues the loop 2 times and creates 2 child processes, P12 with i = 3, P13 with i = 4.

P12 starts with i = 3, continues the loop one time and creates 1 child process, P14 with i = 4.

P3 starts with i = 3, continues the loop one time and creates 1 child process, P15 with i = 4.

2. a) 3 Threads, the main thread and two child threads

b) Six cases:

x = 6, y = 12; t1 executes completely before t2
x = 11, y = 11; t2 executes completely before t1
x = 12, y = 12; t1 executes y = y + 1, t2 runs till completion, and then t1 executes x = y
x = 6, y = 6; t1 reads y, t2 runs till completion, and then t1 continues
x = 6, y = 10; t2 reads y, t1 runs till completion, and then t2 continues
x = 10, y = 10; t2 reads y, t1 reads y and execute y = y +1, t2 executes y = y * 2, and then x=y

Note,

y = y + 1, consists of three assemby instructions :

```
Load R1, y
Add R1, 1, R1
Store R1, y
```

x = y, consists of two assemby instructions :

```
Load R1, y
Store R1, x
```

y = y * 2, consists of three assemby instructions :

```
Load R1, y
Mul R1, 2, R1
Store R1, y
```

3. The major flaw is the violation of **absence of unnecessary delay**. A thread's id must be equal to the turn in order to enter a critical section (CS), otherwise it will spin. When a thread leaves the CS, it changes the turn to the other thread's id (e.g. thread 0 will change the turn to 1, and thread 1 will change the turn to 0). So, this solution requires that the access to the CS has to strictly alternate between thread 0 and thread 1. This is unnecessary.

If only one thread wants to enter the critical section, it may never get to do so, or only get to do it once. For example, if only thread 1 wanted to enter the CS, it would spin infinitely because the turn is initially 0.

4.

```
int x = 0;
entry() {
        int y;

        y  = Inc(x);
        while (x > 1) {
                y = Inc(x);
        }
}
exit() {
        x = 0;
}
```

Note, ignore the possible Integer overflow.
Note, the initial value of x and the while loop condition are linked. There are many possible correct answers

5.

(a) There is no mutual exclusive access to the shared array available[n]. If two threads call allocate() simultaneously. One thread will read available[i] == true and before it can change it to available[i] = false, the other thread reads the same available[i] == true and continues to try and change the value to false. Therefore, both customers are assigned to the same machine.

(b)

The *available* array is initialized to all true, and *nfree* is initialized to NMACHINES. The mutex is initialized to 1.

```
int allocate() /*Returns index of available machine */
{
      wait(nfree); /*Wait until a machine is available */
      wait(mutex); /* lock the critical section */
      for (int i=0; i < NMACHINES; i++) {
            if (available[i]) {
                  available[i] = FALSE;
                  signal(mutex); /*release the lock before return */
                  return i;
            }
      }
}

void release (int machine) /* release machine */
{
      available[machine] = TRUE;
      signal(nfree);
}
```

6.

```
monitor concurrentCompute {
    int a, b, c, d, e
    bool aAvail = false, bAvail = false, dAvail = false, eAvail = false;
    condition aAvailCond, bAvailCond, dAvailCond, eAvailCond;

    readA() {
        cin >>a;
        aAvail = true;
        aAvailCond->signal();
    }

    readB() {
        cin >>b;
        bAvail = true;
        bAvailCond->broadcast();  //wake up all waiting threads for b
    }

    mulC() {
        while (!bAvail) {
            bAvailCond->wait();
        }
        c =  b * 2;
    }

    sumD() {
        while (!bAvail) {
            bAvailCond->wait();
        }

        d =  b + 1;

        dAvail = true;
        dAvailCond->signal();
    }

    sumE() {
        while (!aAvail) {
            aAvailCond->wait();
        }

        while (!dAvail) {
            dAvailCond->wait();
        }

        e =  d - a;

        eAvail = true;
        eAvailCond->signal();
    }

    print() {
        while (!eAvail) {
            eAvailCond->wait();
        }
        cout << a << e;
    }
}
```

7.

(a)

```
Monitor Bank {
        int balance = 0
        cond okToWithdraw;

        void Deposit(int amount) {
                balance = balance + amount;
                okToWithdraw->broadcast();
        }

        void Withdraw(int amount) {

                //wait for funds
                while (amount > balance) {
                        okWithdraw->wait();
                }

                balance = balance – amount;
        }

}
```

(b)

```
Monitor FCFSBank {
        int balance = 0
        cond okToWithdraw;

        int numOfWithdraw = 0;
        cond okToBank;

        void Deposit(int amount) {
                balance = balance + amount;
                okToWithdraw->signal();          //it is also fine,  okToWithdraw->broadcast();
        }

        void Withdraw(int amount) {
                numOfWithdraw++;

                //wait for turn
                if (numOfWithdraw > 1) {       //it cannot be a while loop
                        okToBank->wait();
                }

                //wait for funds
                while (amount > balance) {
                        okToWithdraw->wait();
                }

                balance = balance – amount;

                numOfWithdraw--;
                okToBank->signal();        // signal the next withdraw

        }

}
```