use Adobe Reader to complete

(Type in fields)

Basics4 – Conditional Variables

Student Information

Integrity Policy: All university integrity and class syllabus policies have been followed. I have neither given, nor received, nor have I tolerated others' use of unauthorized aid.

I understand and followed these policies: [✓ Yes] No

Name:	Joey Domino
Date:	11/28/2022

Submission Details

Final ***Changelist*** number:	158557
Verified build:	[✓ Yes] No
Number Tests Passed:	2/4
Required Configurations:	1: { Debug | x86 }

Discussion (What did you learn):

I FORGOT TO UPDATE THIS DANGIT

Verify Builds

- Follow the Piazza procedure on submission
  - Verify your submission compiles and works at the changelist number.
- Verify that only MINIMUM files are submitted
  - No – Generated files
    - *.pdb, *.suo, *.sdf, *.user, *.obj, *.exe, *.log, *.pdb, *.db
    - Anything that is generated by the compiler should not be included
  - No – Generated directories
    - /Debug, /Release, /Log, /ipch, /.vs
- Typical files project files that are required
  - *.sln, *.suo,
  - *.vcxproj, *.vcxproj.filters, *.vcxproj.user
  - *.cpp, *.h
  - CleanMe.bat

Standard Rules

Submit multiple times to Perforce

- Submit your work as you go to perforce several times (at least 5)
  - As soon as you get something working, submit to perforce
  - Have reasonable check-in comments
    - Seriously, I'm checking

Write all programs in cross-platform C++

- Optimize for execution speed and robustness
- Working code doesn't mean full credit

Submission Report

- Fill out the submission Report
  - No report, no grade

Code and project needs to compile and run

- Make sure that your program compiles and runs
  - Warning level ALL …
  - NO Warnings or ERRORS
    - Your code should be squeaky clean.
  - Code needs to work "as-is".
    - No modifications to files or deleting files necessary to compile or run.
  - All your code must compile from perforce with no modifications.
    - Otherwise it's a 0, no exceptions

Project needs to run to completion

- If it crashes for any reason…
    - It will not be graded and you get a 0

Leave Project Settings

- Do NOT change the project or warning level
    - Any changing of level or suppression of warnings is an integrity issue

Leaking Memory

- If the program leaks memory
    - There is a deduction of 20% of grade
- If a class creates an object using new/malloc
    - It is responsible for its deletion
- Any ***MEMORY*** dynamically allocated that isn't freed up is ***LEAKING***
    - Leaking is ***HORRIBLE***, so you lose points

No Debug code or files disabled

- Make sure the program is returned to the original state
    - If you added debug code, please return to original state
- If you disabled file, you need to re-enable the files
    - All files must be active to get credit.
    - Better to lose points for unit tests than to disable and lose all points
- Disable your debug printing otherwise you will lose points

Due Dates

- See Piazza for due date and time
- Submit program perforce in your student directory assignment supplied.
- Fill out your this ***Submission Report*** and commit to perforce
    - ***ONLY*** use Adobe Reader to fill out form, all others will be rejected.
    - Fill out the form and discussion for full credit.

Goals

- Learn
    - Conditional Variables
    - Different types of wait()
    - Notify_one(), Notify_all()

Assignments

1. ***Problem_1 / Problem_2***
   - BACKGROUND
     - ***Producer*** class
       - A producer thread is created from the functor class called Producer.
       - This class sets the shared data, both the value and the count value.
       - Randomized time - it updates the shared data
         a. Value ← updates
         b. Count ← updates (a random pattern)
         c. Complement ← clears the complement
       - After updating it calls ***notify_one()*** on shared conditional variable
     - ***Consumer_X / Consumer_Y*** class
       - Several consumer threads are created from the functor class
       - This functor class you modify.
         a. You ***CANNOT*** add any data as class member data.
         b. You can add local variables to the functor
         c. You can add methods to shared data class for predicates
         d. Follow the comments
            i. You will need to create a condition variable using either ***wait_for()*** or ***wait_until()***
       - Inside the functor…
         a. You need to process/update shared data.
            i. It is updated by the Producer thread at a different durations
         b. You are going to read the shared data
            i. Store value data one's complement → complement field
       - This function cannot sleep this thread directly
         a. Indirectly by using conditional variables correctly
       - You need to use on of the conditional variable ***wait*** functions
         a. ***wait_for***() or ***wait_until***() – look at **comments** that specify which one to use
         b. Guarantee that the correct data is applied to only one object per Producer update
         c. Make sure it leaves the thread correctly
     - ***SharedData*** class
       - This class hold data that is sent from the producer to the consumers
       - Consumer class updates the complement data
       - You cannot add any member data to this class

- You can add methods if you want.

- ACTION
  - The producer updates the data at randomized intervals
  - There are 4 consumers trying to update the shared data
  - Only allow one consumer to update new data
  - Look at the implementation of wait functions to see the internal implementation… understand how and why it locks/unlocks the "lock" aka mutex.
- OUTPUT
  - The output should match this sample:
    - The order the consumer threads are launch may be in a different order
    - The actual consumer thread that updates the shared data may be different than this sample
      a. What matters is that only one consumer thread updates on the producer's update.
         i. ***notify_one()*** – triggers that… but you need to make sure you grab it correctly in the consumer threads
  - See Sample Output.txt

2. ***Problem_3 / Problem_4***
   - BACKGROUND
     - ***Producer*** class
       - A producer thread is created from the functor class called Producer.
       - This class sets the shared data, both the value and the count value.
       - Every 1 second it updates the shared data
         a. Value 🡐 updates
         b. Count 🡐 updates (increments)
         c. Complement 🡐 clears the complement
       - After updating it calls ***notify_all()*** on shared conditional variable
     - ***Consumer_Z / Consumer_W*** class
       - Several consumer threads are created from the functor class
       - This functor class you modify.
         a. You ***CANNOT*** add any data as class member data.
         b. You can add local variables to the functor
         c. You can add methods to shared data class for predicates
         d. Follow the comments
            i. You will need to create a condition variable using either ***wait_for()*** or ***wait_until()***

- Inside the functor…
  a. You need to process/update shared data.
     i. It is updated by the Producer thread at random intervals
  b. You are going to read the shared data
     i. Store value data one's complement → complement field
- This function cannot sleep this thread directly
  a. Indirectly by using conditional variables correctly
- You need to use on of the conditional variable ***wait*** functions
  a. ***wait_for***() or ***wait_until***() – follow the ***comments***
  b. Guarantee that the correct data is applied once to ***EVERY*** object per Producer update
  c. Make sure it leaves the thread correctly

- ***SharedData*** class
  - This class hold data that is sent from the producer to the consumers
  - Consumer class updates the complement data
  - You cannot add any member data to this class
  - You can add methods if you want.

- ACTION
  - The producer updates the data at randomized intervals
  - There are 4 consumers trying to update the shared data
  - Allow all consumer threads to update new data
    - There are 4 consumer threads… so there should be 4 updates
  - Look at the implementation of wait functions to see the internal implementation… understand how and why it locks/unlocks the "lock" aka mutex.
- OUTPUT
  - The output should match this sample:
    - The order the consumer threads are launch may be in a different order
    - The actual consumer thread that updates the shared data may be different than this sample
      a. What matters is that ***ALL*** consumer thread updates on the producer's update.
         i. ***notify_all()*** – triggers that… but you need to make sure you grab it correctly in the consumer threads
  - See Sample Output.txt

3. ***Make sure it builds / runs in Debug configurations***
   - Implement and develop on Debug/x86
   - After that configuration works → verify the configurations:
     - Debug x86

Validation

Simple checklist to make sure that everything is submitted correctly

- Is the project compiling and running without any errors or warnings?
- Does the project run ***ALL*** in all configurations without crashing?
- Is the submission report filled in and submitted to perforce?
- Follow the verification process for perforce
  - Is all the code there and compiles "as-is"?
  - No extra files
- Is the project leaking memory?

Hints

Most assignments will have hints in a section like this.

- Do many little check-ins
  - Iteration is easy and it helps.
  - Perforce is good at it.
- READ the book (chapter 4)
  - Many good ideas in there.
- I had to do a lot of googling and web searching
  - Not make examples out there.
  - Dig into it you'll get it