**Seattle University**

**Spring**

12

**Seattle University 900 Broadway , Seattle, WA 98122**

. .

This report documents the development of the Concur project: the problems it aims to solve, the design and implementation of the system, how it was tested, and an analysis of how well the system meets its requirements.

Graduate Implementation Project

08

**Fall**

Concur: A Testing and Debugging Tool for Concurrent Applications Technical Report

# Table of Contents

Table of Contents 2

1 Abstract 4

2 Introduction 4

3 Discussion of the Project 4

3.1 Design Specification 5

3.1.1 Requirements 5

3.1.2 Use Cases 6

3.1.3 User Interface 7

3.2 Software Implementation 7

4 Analysis of the Results 8

5 Testing 9

5.1 Test Plan 9

5.2 Test Specification 9

5.2.1 FE-01: Random Mode for Thread Selection 10

5.2.2 FE-02: Interactive Mode for Thread Selection 10

5.2.3 FE-03: Log Trail 10

5.2.4 FE-04: Thread, Lock, and Semaphore Status 10

5.2.5 FE-05: Wrapper 10

5.2.6 FE-06: Deadlock Detection 11

5.2.7 FE-07: Non-Deadlock Error Detection 11

5.2.8 FE-08: Reproduce Failed Test Run 11

5.2.9 FE-09: Startup 11

5.3 Test Cases 11

5.3.1 FE-01: Random Mode for Thread Selection Test Cases 11

5.3.2 FE-02: Interactive Mode for Thread Selection Test Cases 11

5.3.3 FE-03: Log Trail Test Cases 12

5.3.4 FE-04: Thread, Lock, and Semaphore Status Test Cases 12

5.3.5 FE-05: Wrapper Test Cases 12

5.3.6 FE-06: Deadlock Detection 14

5.3.7 FE-07: Non-Deadlock Error Detection 15

5.3.8 FE-08: Reproduce Failed Test Run Test Cases 16

5.3.9 FE-09: Startup Test Cases 16

6 User Documentation 17

7 Deployment and Maintenance Guidelines 18

8 Dependencies on Other Tools 18

Appendix A. Title 19

Appendix B. Title 19

**List of Figures**

No table of figures entries found. **In your document, select the words to include in the table of contents, and then in the Formatting Palette under Styles, click a heading style. Repeat for each heading that you want to include, and then insert the table of contents in your document. You can also create a table of contents by clicking the Create with Manual Formatting option and then type the entries manually.**

**List of Tables**

**No table of figures entries found.**  
In your document, select the words to include in the table of contents, and then on the Home tab, under Styles, click a heading style. Repeat for each heading that you want to include, and then insert the table of contents in your document. To manually create a table of contents, on the Document Elements tab, under Table of Contents, point to a style and then click the down arrow button. Click one of the styles under Manual Table of Contents, and then type the entries manually.

# Abstract

This report documents the development of the Concur test framework, which was completed over two academic quarters as a graduate implementation project. The beginning of this report provides a discussion of the project including the design specification and software implementation. Following the project discussion is the presentation, discussion, and analysis of the results in which we discuss how the end system faired with our design. Next an analysis of errors is discussed as well as the approach we took to determine the root cause of any errors. A test section that includes the test plan, specification, and cases used to test our system is provided. The summary and conclusion sections provide a look back at the main points of the report. Finally, the Concur test framework source code is provided in the Appendices.

# Introduction

The primary goal of the graduate software implementation project was to design and implement software that aims to address problems. The problems that Concur was designed to address are those related to testing and debugging multi-threaded applications.

The key problem is that students do not have sufficient resources to adequately determine if their multithreaded program is free from deadlock and/or race conditions. Testing alone is insufficient because many conditions require specific schedules that are not exercised. This leads to the need of a tool that assists students in finding and removing synchronization bugs.

For this project I built Concur, which is a testing and debugging tool for multithreaded programs involving synchronization. Concur was built as a new system making use of existing pieces. There are two primary goals of Concur that both aim at helping students with little or no experience with multithreading. The first goal is to allow a student to test his or her multithreaded program for correctness (i.e. detect bugs). The second goal is to help a student debug the program when an error is present. Concur achieves these goals by providing an environment in which the user has more control over thread scheduling, displaying the current state of each thread, lock and semaphore, detecting synchronization errors such as deadlocks, and reproducing test results. Teachers will also benefit from Concur when used as an instructional aid or for evaluating multithreaded programs.

# Discussion of the Project

Several high-level requirements have been identified for the system. The functional requirements include allowing control over thread scheduling (through random and interactive modes), capturing programming errors related to multithreading and synchronization, making certain information available to facilitate debugging efforts (e.g. the current state of each thread, lock, and semaphore and information about the failure). The nonfunctional requirements include usability, maintainability, and extensibility.

## Design Specification

### Requirements

Several key functional and non-functional requirements have been identified for Concur.

Functional Requirements:

|  |  |  |
| --- | --- | --- |
| ID | Title | Description |
| F01 | Random mode for scheduling | Concur shall support a random mode for scheduling in which threads are automatically and randomly scheduled. |
| F02 | Interactive mode for scheduling | Concur shall support an interactive mode for scheduling in which the user will select which thread to run next. |
| F03 | Failure trail | Concur shall provide information about the failure. |
| F04 | Thread, lock and semaphore status | Concur shall provide current state of each thread, lock, and semaphore when in interactive mode. |

Non-Functional Requirements:

|  |  |  |
| --- | --- | --- |
| ID | Title | Description |
| NF01 | Usability | Concur shall be usable for the intended users. |
| NF02 | Maintainability | Concur shall be maintainable |
| NF03 | Extensibility | Concur shall be extensible |

Requirements F01 and F02 require Concur to provide two scheduling modes that grant the user varying degrees of control over how the threads shall be scheduled for execution.

For requirement F01: Random mode for scheduling, the order of thread execution will be determined randomly and automatically. The benefit of this method is that it requires no user intervention. The disadvantage, however, is that even after numerous random executions not all of the interesting cases may have been explored.

For requirement F02: Interactive mode for scheduling, Concur shall allow the user to specify the schedule of threads. A positive aspect of this method is that the student is forced to think about synchronization and testing. A drawback, however, is that continuously choosing which thread to execute next can be tedious and only a few schedules would get run.

Requirement F03: Failure trail requires Concur to output information regarding an encountered failure. In random mode, Concur will provide some information about the failure including the schedule that exercised the bug.

Requirement F04: Thread, lock, and semaphore status requires Concur to provide current status of each thread, lock, and semaphore.

The features developed to satisfy requirements F03 and F04 help the user debug and capture most of the programming mistakes related to multithreading and synchronization made by the student. In random mode, the failure information gained from requirement F03 will inform the user that an error occurred. The error could be dependent on the problem, however, such as the restrictions in the unisex bathroom problem. The use of either assertions or separate checkers could help address these cases. Concur could allow a deadlock to manifest itself by letting the program “hang”, but the user would need to distinguish between a long running execution versus one that hangs. For random modes, it suffices to run an evaluation to see if it is effective or not in finding errors. For interactive modes, trying to determine if the error can be reproduced would satisfy requirement F03.

Although Concur in random mode exposes errors, it does not necessarily uncover what failure occurred and for what reason. The user will need to do additional investigation in interactive mode to determine what type of failure occurred (deadlock or race condition) and why the failure transpired. In interactive mode, the user can step through the program execution following the same thread schedule that exercised the bug. By examining the current state of each thread, lock, and semaphore leading up to the failure, the user should gain a clearer understanding with regards to what caused the error. The user can then make necessary updates to the code and rerun the failed schedule until the error has been resolved. The user would repeat this debugging process until the program is error free.

The Concur must also satisfy the quality requirements of usability, maintainability, and extensibility (requirements NF01, NF02, and NF03, respectively). It must be usable for student programmers that have some familiarity with Linux but are new to synchronization. The tool must also be extensible to a more systematic approach and should be able to support different synchronization problems with their checkers.

### Use Cases

The primary user of Concur is a student who is beginning to learn multithreaded programming and synchronization. The student can use Concur to test the program for errors, using its controlled environment to increase test case coverage and hence increase the chances of detecting bugs. If an error is detected, the student can then use Concur to debug the program. Concur will provide some information about the failure including the schedule that exercised the bug. With this information along with the ability to choose which threads to execute, the student will be equipped to reproduce the execution that caused the error. The ability to reproduce the failed execution is a powerful tool in debugging a multithreaded application that the student typically wouldn’t have due to the nondeterministic nature of the operating system’s scheduler.

The secondary user of Concur is a teacher who can use the tool for demonstration and for evaluation. A teacher can use Concur in class to demonstrate how deadlock and/or race conditions can occur in a program. The tool can also be used to discover programming errors while evaluating students’ programming assignments.

TODO: Insert Use Case Diagram and Use Cases

### User Interface

The user will run Concur via a terminal window. The user will provide a program file that adheres to a certain specification that will be tested by the framework. The user can execute Concur that runs once through the multithreaded application, or he can use the wrapper tool that automatically executes Concur multiple, sequential times. Concur allows thread selection (e.g. the next thread to run, the next waiting thread to acquire a lock, or the next waiting thread to use a semaphore) to be done either interactively by the user or automatically and randomly.

## Software Implementation

Input: Concur accepts a multithreaded program to be tested as input along with the following control parameters: number of threads, number of rounds, scheduler mode, logging mode, and random seed.

Components: Concur consists of the following main components:

* Instrumentation tool: The instrumentation tool takes a copy of the input program and insert calls to the thread “proxy” scheduler throughout the test program. This is an existing component that was provided by the faculty advisor.
* Scheduler: A “proxy” scheduler exists between the instrumented code and the operating system’s scheduler. The “proxy” scheduler will make calls to the OS scheduler on behalf of the instrumented program. The scheduler will be knowledgeable of the program’s multithreading and synchronization information such as the number and states of threads, and number and states of locks and semaphores. This set of information and control allows the tool to transfer thread execution control to the user.
* Wrapper: A wrapper or shell component runs multiple sequential executions of Concur over the input program.
* Logging: Concur outputs any errors encountered and the schedule of threads executed during the test run. For interactive mode, the system will prompt the user to select the thread to run, acquire a lock, and/or use a semaphore during various points of test execution. The system will also output current thread, lock, and semaphore states to the user.

# Analysis of the Results

# Testing

This section includes the test plan, test specification, and test cases used to test Concur.

## Test Plan

The following describes the test of the Concur test framework. In each execution of Concur, a single user-provided sections.c file will be tested with the following control parameters set by the user or inferred by another control parameter: number of test runs, number of threads, number of rounds, scheduler mode, and logging mode.

To ensure operation of the system according to the initial requirements and design specification, the following items must be tested:

1. Ability to run in random scheduler mode
2. Ability to run in interactive scheduler mode
3. Ability to view thread, lock, and semaphore status
4. Ability to log results
5. Ability to detect deadlocks
6. Ability to run sequential test runs in random mode

## Test Specification

This section provides more detail regarding the function to be tested and the input to be used.

Inputs:

* sections.c file: This file is to be provided by the user and must include the following functions to be called by main.c:
  + initGlobals(int id): initialize variables global to the application including locks and semaphores
  + entrySection(int id)
  + criticalSection(int id)
  + exitSection(int id)
  + remainderSection(int id).
* Number of threads: This is a positive integer that indicates how many child threads to execute within the program.
* Number of rounds: This is a positive integer that indicates how many rounds each thread will run. Each round consists of a single execution of each of the entry, critical, exit, and remainder section functions.
* Scheduler mode: This is one of the following strings: “random” or “interactive”. This input is not case sensitive. If scheduler mode is “random”, thread selection will be performed randomly. If scheduler mode is interactive, thread selection will be performed through user prompts. Scheduler mode is set to “random” when running via the Wrapper.
* Logging mode: This is one of the following strings: “on” or “off”. This is not case sensitive. If logging mode is on, an output file will be produced and the test run status will be written to it. If logging mode is on, no log file will be created. Logging mode is set to “off” when running via the wrapper.
* Number of runs (applicable only with the Wrapper): This is a positive integer indicating how many Concur executions to run.
* via Concur
  + If any of the Concur inputs are invalid, Concur shall return an error message to the console and exit.

### FE-01: Random Mode for Thread Selection

If scheduler mode is “random”, thread selection shall be done randomly.

If random seed is -1 and scheduler mode is random, Concur shall automatically generate the random seed.

### FE-02: Interactive Mode for Thread Selection

If scheduler mode is “interactive”, Concur shall prompt the user for thread selection. The random seed parameter is ignored.

### FE-03: Log Trail

If logging mode is on, Concur shall open a new output file and log status to it during program execution. Concur shall safely close the output file upon exiting the program.

If logging mode is off, no output file shall be created.

### FE-04: Thread, Lock, and Semaphore Status

If logging mode is “on”, thread, lock, and semaphore status shall be output to log file at each call to invokeScheduler(…).

### FE-05: Wrapper

FE-05.1: If any of the Wrapper inputs are invalid, the Wrapper shall return an error message to the console and exit.

FE-05.2: The Wrapper will continue to execute additional Concur test runs provided each test run completes without error.

FE-05.3: If Concur returns an error for the current test run, the Wrapper shall output an error message to the screen and exit. No additional test runs will be performed once a failed test run is encountered.

FE-05.4: If Concur hangs due to a non-deadlock error, the Wrapper will also hang. The user will have to kill the process (CTRL+C).

### FE-06: Deadlock Detection

If all active threads are blocked waiting for a lock or semaphore, Concur shall output an error message indicating that a deadlock has occurred and exit the application with an error exit status.

### FE-07: Non-Deadlock Error Detection

If Concur encounters a non-deadlock error, the user shall be able to examine the log trail and determine at what point during execution the error was encountered.

If all threads have completed without error, Concur shall return with a success status.

### FE-08: Reproduce Failed Test Run

If Concur receives inputs identical to control parameters of a failed test run, Concur shall reproduce the same failure.

### FE-09: Startup

Upon startup, Concur shall output to the screen the values of the following control parameters used for the run: number of threads, number of rounds, scheduler mode, logging mode, and random seed value.

## Test Cases

The following test cases identify how each component of the system identified in the test plan and quantified in the test specification is to be tested.

### FE-01: Random Mode for Thread Selection Test Cases

|  |  |
| --- | --- |
| **ID:** | **FE­01\_TC1** |
| TCS Name: |  |
| Description: | This test will determine |
| Expected Outcome: |  |
| Status: |  |

### FE-02: Interactive Mode for Thread Selection Test Cases

|  |  |
| --- | --- |
| **ID:** | **FE­02\_TC1** |
| TCS Name: |  |
| Description: | This test will determine |
| Expected Outcome: |  |
| Status: |  |

### FE-03: Log Trail Test Cases

|  |  |
| --- | --- |
| **ID:** | **FE­03\_TC1** |
| TCS Name: |  |
| Description: | This test will determine |
| Expected Outcome: |  |
| Status: |  |

### FE-04: Thread, Lock, and Semaphore Status Test Cases

|  |  |
| --- | --- |
| **ID:** | **FE­04\_TC1** |
| TCS Name: | Thread, Lock, and Semaphore Status is Outputted |
| Description: | This test will determine whether thread, lock and semaphore status is outputted to the log file with the required fields.  Input:   * sections file: sections.1.c * ./concur.synch.exe 5 2 RANDOM ON 1338753699 * number of threads = 5 * number of rounds = 2 * schedule mode = RANDOM * log mode = ON * seed = 1338753699 |
| Expected Outcome: | Thread, lock, and semaphore status will be outputted to the log file with the required fields. |
| Status: | Success: The status was outputted to the log file. |

### FE-05: Wrapper Test Cases

|  |  |
| --- | --- |
| **ID:** | **FE­05\_TC1** |
| TCS Name: | Wrapper Receives Invalid Input |
| Description: | This test will determine the Wrapper’s response to invalid input parameters. |
| Expected Outcome: | The Wrapper will output an error message and exit. No test runs will be executed. |
| Status: | TBC |

|  |  |
| --- | --- |
| **ID:** | **FE­05\_TC2** |
| TCS Name: | Wrapper with successful test runs. |
| Description: | This test will determine  if the wrapper will continue to run the subsequent test run if the current test run completes without error.  Input to Wrapper:   * Sections file: sections.3.c * number of test runs = 10 * number of threads = 5 * number of rounds = 2 |
| Expected Outcome: | SUCCESS |
| Status: | Success. All test runs were completed without error.  Console output from Wrapper:  [paltingr@css2 concur]$ ./run-concur.py --runs=10 --threads=5 --rounds=2  ./concur.synch.exe 5 2 RANDOM OFF  Run 1:  -- numThreads=5, numRounds=2, schedMode=RANDOM, logMode=OFF, seed=1338707229  Run 2:  -- numThreads=5, numRounds=2, schedMode=RANDOM, logMode=OFF, seed=1338707238  Run 3:  -- numThreads=5, numRounds=2, schedMode=RANDOM, logMode=OFF, seed=1338707246  Run 4:  -- numThreads=5, numRounds=2, schedMode=RANDOM, logMode=OFF, seed=1338707255  Run 5:  -- numThreads=5, numRounds=2, schedMode=RANDOM, logMode=OFF, seed=1338707266  Run 6:  -- numThreads=5, numRounds=2, schedMode=RANDOM, logMode=OFF, seed=1338707277  Run 7:  -- numThreads=5, numRounds=2, schedMode=RANDOM, logMode=OFF, seed=1338707287  Run 8:  -- numThreads=5, numRounds=2, schedMode=RANDOM, logMode=OFF, seed=1338707296  Run 9:  -- numThreads=5, numRounds=2, schedMode=RANDOM, logMode=OFF, seed=1338707307  Run 10:  -- numThreads=5, numRounds=2, schedMode=RANDOM, logMode=OFF, seed=1338707316  ----All tests PASSED |

|  |  |
| --- | --- |
| **ID:** | **FE­05\_TC3** |
| TCS Name: | Wrapper encounters deadlock. |
| Description: | This test will determine how the Wrapper responds when one of the test runs encounters a deadlock error (i.e. Concur exits with an error status).  Input:   * sections file: sections.1.c * ./run-concur.py --runs=10 --threads=5 --rounds=2 * number of runs = 10 * number of threads = 5 * number of rounds = 2 |
| Expected Outcome: | The Wrapper will output an error message and exit. No additional test runs will be executed. |
| Status: | Success. Deadlock encountered. Concur returned with an error status code. Wrapper output a message that the test run failed and the script was exited.  [paltingr@css2 concur]$ ./run-concur.py --runs=10 --threads=5 --rounds=2  ./concur.synch.exe 5 2 RANDOM OFF  Run 1:  -- numThreads=5, numRounds=2, schedMode=RANDOM, logMode=OFF, seed=1338763589  ----Test FAILED at round 1. |

|  |  |
| --- | --- |
| **ID:** | **FE­05\_TC4** |
| TCS Name: | Wrapper Encounters Non-Deadlock Error |
| Description: | This test will determine how the Wrapper responds when one of the test runs fails due to a non-deadlock error (Concur hangs and fails to return).  The input was kept simple for this test initially to verify that sections could pass for a single thread.  Input:   * sections file: sections.2.c * number of test runs = 1 * number of threads = 1 * number of rounds = 1 |
| Expected Outcome: | If Concur hangs and fails to return, the Wrapper will also hang. The user will have to kill the process to exit. |
| Status: | Success. During the first and only test run, Concur failed to return after waiting approximately one minute. The user killed the process to exit. |

### FE-06: Deadlock Detection

|  |  |
| --- | --- |
| **ID:** | **FE­03\_TC1** |
| TCS Name: | Deadlock Detection |
| Description: | This test will determine |
| Expected Outcome: | This test will determine whether Concur can detect deadlock within a program.  Inputs:   * sections file: sections.1.c * ./concur.synch.exe 5 2 RANDOM ON 1338753699 * number of threads = 5 * number of rounds = 2 * schedule mode = RANDOM * log mode = ON * seed = 1338753699 |
| Status: |  |

### FE-07: Non-Deadlock Error Detection

|  |  |
| --- | --- |
| **ID:** | **FE­07\_TC1** |
| TCS Name: | Non-Deadlock Error Detection |
| Description: | This test will determine whether the user will be able to determine at what point during program execution a non-deadlock error occurred.  Input:   * sections file: sections.2.c * number of threads = 1 * number of rounds = 1 * scheduler mode = RANDOM * log mode = ON |
| Expected Outcome: | If Concur hangs, the last log trail entries should provide enough information for the user to determine at what point the program hung. In particular, the sections.c line number shall be provided. |
| Status: | Success. After Concur hung, I examined the last entry in the log trail and saw that the line corresponded to a call to function personEnterRestroom(id, isFemale). From examining the first line of this function, it is evident that there is an infinite loop error when isFemale=true (isFemaleUsingBathroom is initialized to false):  while( isFemale != isFemaleUsingBathroom );  Concur output:  [paltingr@css2 concur]$ ./concur.synch.exe 1 1 RANDOM ON 1338742293  output/output-20120603\_10:38:56.txt  -- numThreads=1, numRounds=1, schedMode=RANDOM, logMode=ON, seed=1338742293  [paltingr@css2 concur]$ |

### FE-08: Reproduce Failed Test Run Test Cases

|  |  |
| --- | --- |
| **ID:** | **FE­08\_TC1** |
| TCS Name: |  |
| Description: | This test will determine |
| Expected Outcome: |  |
| Status: |  |

### FE-09: Startup Test Cases

|  |  |
| --- | --- |
| **ID:** | **FE­09\_TC1** |
| TCS Name: |  |
| Description: | This test will determine |
| Expected Outcome: |  |
| Status: |  |

# User Documentation

# Deployment and Maintenance Guidelines

# Dependencies on Other Tools

# Appendix A. Title

# Appendix B. Title