Taylor & Francis
Taylor & Francis Group

Automatika

Journal for Control, Measurement, Electronics, Computing and Communications

ISSN: 0005-1144 (Print) 1848-3380 (Online) Journal homepage: https://www.tandfonline.com/loi/taut20

Application of social game context to teaching mutual exclusion

Miroslav Popović, Klemo Vladimir & Marin Šilić

To cite this article: Miroslav Popović, Klemo Vladimir & Marin Šilić (2018) Application of social game context to teaching mutual exclusion, Automatika, 59:2, 208-219, DOI: 10.1080/00051144.2018.1522462

To link to this article: https://doi.org/10.1080/00051144.2018.1522462

Published online: 19 Sep 2018.

AUTOMATIKA
2018, VOL. 59, NO. 2, 208–219
https://doi.org/10.1080/00051144.2018.1522462

REGULAR PAPER

OPEN ACCESS Check for updates

Application of social game context to teaching mutual exclusion

Miroslav Popović, Klemo Vladimir and Marin Šilić

Department ZEMRIS, Faculty of Electrical Engineering and Computing, University of Zagreb, Zagreb, Croatia

ABSTRACT

Mutual exclusion mechanisms, like semaphore and monitor, are fundamental tools used by software engineers to solve the race condition problem, ensure barrier, and achieve other workflow patterns. Introductory teachings on how parallel and concurrent processes compete over shared resources have the underlying working principles of the operating system and computer architecture as a starting point for learning the mutual exclusion concepts. Conventional teaching method focuses on lectures and solving race condition problem with counting semaphore in C programming language. Before applying conventional teaching method, we advocate the introduction of a social game scenario in teaching basic concepts of workers concurrently competing over a shared resource. We also introduce a simplified mutual exclusion assignment in which the implementation complexity is reduced by application of a specially designed graphical mechanism for mutual exclusion. Compared to a conventional method, the proposed experimental teaching method has a 15% higher success rate in solving race condition problem in C programming language. Regardless of additional steps introduced to make students familiar with the concepts of mutual exclusion, the experimental method is slightly advantageous when median time-on-task results are compared.

ARTICLE HISTORY
Received 2 June 2017
Accepted 6 September 2018

KEYWORDS
Education; concurrency; mutual exclusion; social game

Introduction

The most recent software engineering industry trends are witnessing the increase usage of various cloud platforms, sophisticated server side technologies and multicore processing applications. As part of these trends, there is an ubiquitous need for software engineers strongly proficient in parallel and distributed computing (PDC) which is emphasized in the hiring process. These demands are well recognized and addressed by curriculum recommendations which prescribe the body of parallel and distributed computing knowledge for computer science undergraduates [1,2].

Conventional teaching methods address topics from PDC through lectures and practical assignments. Lectures are commonly complemented with additional resources which prepare students for the assignment. With minor differences across educational institutions, it is generally agreed that lectures and additional resources need to provide numerous examples of solutions to familiar problems on the PDC topics.

While the importance of numerous well-elaborated examples is unquestionable, the paper researches importance and power of elementary examples which reveal technology independent PDC concepts on real-life metaphors. As a case study, the paper focuses on teaching race condition problem and mutual exclusion techniques, which are considered demanding topics for the undergraduates. In our teaching experience, students find it most difficult to make the first step towards race condition problem which might be related to the poor understanding of parallelism and difficulty to think in terms of parallel and concurrent processes.

The experimental method for teaching PDC concepts on race condition problem is proposed in the paper. In addition to the techniques covered by conventional teaching methods, an experimental method is designed with elements of a social game in which two additional micro assignments for teaching race condition problem are introduced. The micro-assignments are targeted to provide an early introduction on race condition problem and concentrate on the strengthening students' ability to think in terms of parallel and concurrent processes. In the scope of micro assignments, the method avoids having root premises in the underlying working principles of the operating system and the computer architecture. It advocates teaching of basic PDC concepts by relying on the technique of kinesthetic learning by doing (Sivilotti and Pike [3]) and utilizes a specially designed graphical mechanism. We argue that students should have the basic concepts demonstrated using a trivial technology that does not overshadow the concepts with the implementation.

A small-scale experiment evaluates experimental and conventional teaching method on a group of 25 participants who are computer science students enrolled in the Operating Systems course[1]. The

CONTACT Miroslav Popović popovic.miroslav@gmail.com Department ZEMRIS, Faculty of Electrical Engineering and Computing, University of Zagreb, Unska 3, Zagreb 10000, Croatia

proposed experimental teaching method has a 15% higher success rate than the conventional method in solving race condition problem with counting semaphore in C programming language. The experimental method is also slightly advantageous when median time-on-task results are compared.

After presenting related work, the paper describes the conventional method. The experimental method is described next. The evaluation and analysis of the conventional and experimental method are described and followed by a conclusion.

Related work

Shene and Carr [4] have stated early observations on the importance and problems while teaching multithreaded programming. They have found that the concept of race conditions, use of critical sections, and mutual exclusion mechanisms proposed by Dijkstra [5], Lamport [6], and Hoare [7] are far more difficult than anticipated. Shene and Carr [4] also point out that without a deeper understanding of issues which are summarized as the five usual problems students encounter, students will not be able to write decent multithreaded programs.

Cloud platforms, server-side technologies, and multi-core trends from the industry, reported by Lee [8], emphasize demands for software engineers that are strongly proficient in mutual exclusion techniques, multitasking and multiprocessing. These demands are well recognized and addressed by curriculum recommendations which prescribe the body of parallel and distributed computing knowledge for computer science undergraduates [1,2]. According to recommended curricula, more time should be devoted to teaching PDC topics.

Report from Arroyo [9] states that changes are being made across educational institutions to align with the prescribed body of PDC knowledge. A few new teaching approaches have been developed recently in order to effectively transfer the prescribed body of PDC knowledge. Adams, Brown, and Shoop [10] recommendations are to provide extensive number of exemplar applications for parallel programming patterns, where exemplar applications demonstrate PDC topics and techniques in solving some (more or less) realistic problem. Von Praun [11] refers to Scott [12] in support of proposing a course organized to introduce parallelism concepts from higher to lower level of abstraction. Gregg et al [13] research the idea of teaching parallel programming before sequential and report on experimental parallel programming course taught to nine- and ten-year-olds.

To ease the difficulty in teaching and learning multithreaded programming, various visualization tools have been proposed. Visualization proposed by Malnati et al [14] is based on recording traces of thread execution. Thread creation events and events of acquiring and releasing locks are recorded and afterwards visualized in the form of a sequence diagram which can help students to better understand the behavior of a multithreaded program. Similar visualization tool proposed by Carr et al [15] provides visualization support for more synchronization primitives, including semaphores, monitors, and channels. In addition to displaying sequence of executions of synchronization primitives, it also provides special views for visualizing suspended threads for each synchronization mechanism and displaying content of messages exchanged over channels. Manickam and Aravind [16] use animation to demonstrate execution of seven mutual exclusion algorithms which meet given criteria of being simple and elegant. State of execution can be observed in each step of the execution. ConEE [17] is a visualization tool and code validator that checks the code for deadlocks and race conditions. The tool validates programs written in a simple language that supports operations with semaphore, mutex, and barrier. ConEE is limited to short programs with a small number of threads, its objective is to help students grasp the complexity of writing concurrent programs. Synchron-ITS system [18] presents a valuable interactive tutoring tool, but is limited in providing a development environment for synchronization of worker tasks. Instead, its functionality focuses on demonstrating the main concepts of shared memory and synchronization visually and interactively.

Noticeably different approach by Sivilotti and Pike [3] advocates teaching PDC topics by applying kinesthetic learning, since it is a powerful and ubiquitous learning style that resonates with many students across all disciplines and levels of education. Kinesthetic learning is a learning style in which students learn by actively carrying out physical activities by standing, walking, talking, pointing, or working with props rather than by passively listening to lectures. Sivilotti and Pike [3] consider courses on distributed computing are uniquely suited to exploiting this learning technique and present a collection of kinesthetic learning activities for a senior undergraduate or graduate-level course on distributed systems.

The concept of gamifying learning experiences is generally oriented on improving motivation and engagement [19,20] in order to achieve better learning performance and, as such, it may apply to PDC. Research examining the implications of gamification is still scarce and guidelines for well-designed gamification learning need to be more deeply researched. Attalia and Arieli-Attalib [21] and Hanus and Fox [22] show gamification can have a negative impact on learning experiences. Positive effects of gamification have been confirmed in [23] when competing with friends and when gamification provides instructions and feedback [24]. In [25] authors have reported on positive and

negative implications of anonymity of group identity when game-based learning is organized in groups. Performance of students with high prior knowledge was positively influenced by the anonimity of group identity, while it was negatively influenced by the salience of group identity. On the contrary, the performance of students with low prior knowledge was negatively influenced by anonimity of group identity, while it was positively influenced by salience of group identity. In [26] authors suggest caution while introducing competition element which can negatively reflect on overall performance, cooperation, and problem-solving, and tends to have a positive relationship with cheating. Caution is also suggested by Christy and Fox [27] concerning the use of leaderboards and social comparison, which is a common element in game based learning. Using leaderboards can result in both, upward and downward comparisons, having implications on academic performance.

Conventional method

The goal of the conventional teaching method is to educate participants and prepare them for successful completion of individual practical assignment where mutual exclusion is achieved by using counting semaphore in C programming language. In the conventional method, the participants are educated through official lectures from an educational institution and are typically allowed to use all the available resources to help them learn how to solve race condition problem from the assignment. In the conventional method which is applied in our research, participants are recruited from students enrolled in the mandatory Operating Systems course at the University of Zagreb, Faculty of Electrical Engineering and Computing. Within the course, participants receive educational support from the faculty, including lectures on the related topics, recommended reading, and online resources. Performance of participants educated through the conventional teaching method is evaluated on a practical assignment on race condition problem. Figure 1 demonstrates the flow of the experiment in the conventional teaching method. As shown in the figure, before the assignment in C language, all participants are introduced to the assignment details.

The participants are given a sample C program in which a race condition problem occurs when program is executed. The sample pseudocode of the program is shown in Table 1. Critical region of the code between read and write to the data source (lines 2–5) is not protected with any mechanism. Although participants are selected to already possess prerequisite computer science knowledge, they are first introduced to all aspects of the program, including the creation of multiple processes and shared memory, and how the race condition problem occurs. Next, the participants are required

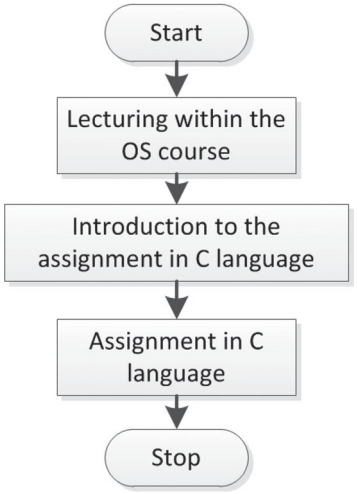

Figure 1. Flowchart diagram of the conventional teaching method.

Table 1. Pseudocode of an individual assignment in the conventional teaching method.

1	while has next income {
2	balance = read balance from the data source;
3	income = read next income;
4	balance = balance + income;
5	store balance to the data source;
6	}

to achieve mutual exclusion by modifying the sample program to use counting semaphore.

The sample program consists of three processes. Each process is assigned an array of small positive integers. All processes have access to a shared memory segment which is initialized to zero. For each element of the array, the process reads the value written in the shared memory, adds the value of the array element, and writes the result of the sum to the shared memory.

Modification of the sample program consists of a call to a primitive for the creation of the counting semaphore, setting of access rights for the semaphore, and defining data structures which are passed as parameters in function calls to the semaphore. Each process needs to make a function call which executes the operation *acquire* semaphore before entering the critical section. After the critical section executes, the process needs to make a function call for executing the operation *release* semaphore. The critical section needs to be as short as possible.

Participants independently self-record the time they spend learning how to solve the problem. Therefore, participants are specially prepared for writing notes and directed to write down the time at the beginning and end of each activity; activities including lectures, programming, reading, looking for resources. The time spent on looking for resources, like browsing the internet, is excluded from the final result. Reports from participants are anonymized to eliminate the motivation for cheating. Regardless of their achievement on

the assignment, participants receive a full score on the practical assignment for the mutual exclusion part of the course.

Experimental method

In addition to teaching steps of the conventional method, the experimental method introduces two additional practical assignments, each preceded by an introductory step. The flow of the experiment is shown in Figure 2. All the additional steps of the experimental method are introduced as a social game. The scenario which describes the assignments is put in the context of collaborative team-work in order to engage students to physically perform mutual exclusion activities and personally experience the race condition problem. A game context is subtly used in the scenario to give meaning to the assignments and make them more fun and enjoyable. Gamification elements such as score points, badges, and leaderboards are considered superficial and misleading from the main non-game objective and therefore are avoided. Participants are required to play a game in which they are assigned roles of accountants and receive imaginary accounting tasks. The tasks make participants compete over a shared resource.

A specially designed mechanism is used in the additional steps in order to avoid having root premises in the underlying working principles of the operating system and the computer architecture. The mechanism is designed to allow the teaching of basic concepts demonstrated within an environment that does not overshadow the concepts with the implementation.

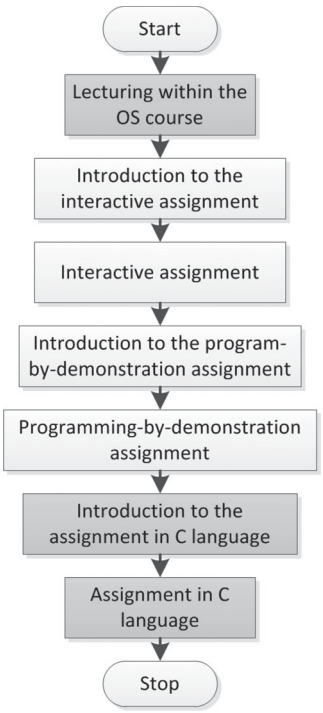

Figure 2. Flowchart diagram of the experimental teaching method.

Geppeto [28,29] is an example of such an environment that is based on a service-oriented architecture and features a widget-oriented programming by demonstration paradigm. In the interactive assignment in the experiment, the graphical mechanism called *Ordered Customer Service* is used for enforcing mutual exclusion. The programming-by-demonstration assignment uses Geppeto programming paradigm to program mutual exclusion over the graphical interfaces of the mechanism used in the previous assignment.

The interactive and programming-by-demonstration assignemnt are performed in a supervised environment in which performance of the participants is controlled and evaluated by the coordinator. Participants are instructed and supervised to precisely measure the time while their efforts are invested in solving the assignment. After stopping the timer, participants report to the coordinator and the coordinator evaluates solutions of each participant.

Introduction to the interactive assignment

A special scenario is designed to introduce participants to a race condition problem that requires mutual exclusion. It is designed with a requirement to place every participant in a situation in which participant personally experiences side effects of unresolved race condition problem. A shared resource is the amount of income on the account of a virtual company. Unprotected *read* and *write* operations to the shared account are exposed in the form of a widget shown in Figure 3(a). Each participant appends its daily income to the shared account. Income appending consists of three consecutive operations. The first operation is a click on the *Refresh* button in order to display the most recent value of company's income. The second operation is an input of the value of a new income to the textual field – the new value is calculated by summing up the most recently read company's income to the personal daily income which participant receives from the coordinator of the experiment. The last operation is a click on the *Store* button which stores the new value to the company's account.

At the beginning of the experiment, participants are grouped in groups of two or three and the sequence of positive numbers representing a sequence of daily incomes is given to each participant. Upon each coordinator's signal of an end of the day, each participant appends one given income to the company's account using the *Account* widget. After participants append all daily incomes, the amount of income of the company is expected to differ from the sum of all daily incomes of all participants in the group. The reason for such result is explained to participants by demonstrating a case where two participants simultaneously perform income appending operations. It is explained that the three income appending operations need to

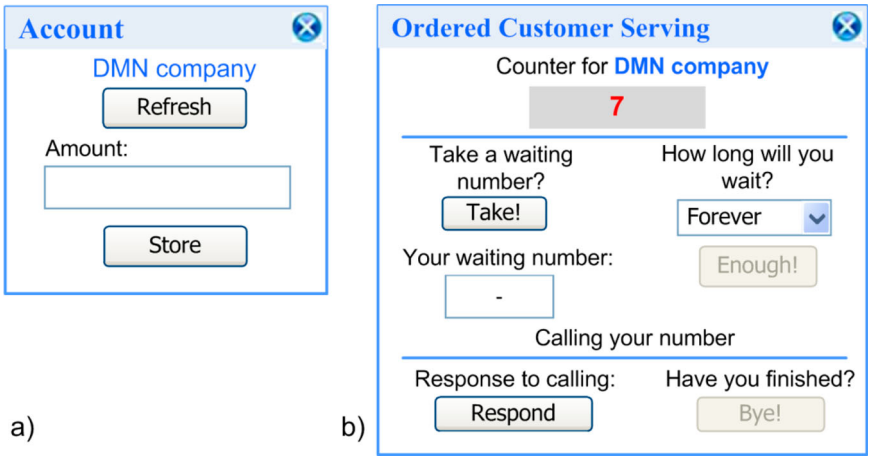

Figure 3. Widgets used in the mutual exclusion scenario: (a) The Account widget for appending of daily income to the company's account; (b) Ordered Customer Serving widget for enforcing mutual exclusion.

execute atomically without other participant's operations interfering. Participants are expected to become aware of the meaning and purpose of mutual exclusion.

In the next attempt, participants are instructed to establish mutual exclusion and complete the same task with the correct final amount of income on the company's account. Participants are directed to establish signaling of performed income appending operations, which signals that the next selected participant can perform income appending. Verbal or any kind of sign communication is allowed for the sole purpose of establishing mutual exclusion protocol.

Interactive assignment

The previously introduced scenario of income appending is modified so that participants are prohibited to engage in any kind of personal communication. The mechanism *Ordered Customer Serving* shown in Figure 3(b) is exposed to participants as the only means of achieving mutual exclusion. The recommendation engine of Geppeto [30] uses existing knowledge of building composite applications with income appending widget to assists participants in discovering and adding the mechanism into the development environment.

Participants need to learn how to properly apply the *Ordered Customer Serving* widget for mutual exclusion of income appending actions that are performed manually using the widget in Figure 3(a). The correct usage of the widget is shown in the form of a manual depicted in Figure 4. The user manual comprises the textual description next to the four-step illustration of the mechanism usage. Mutual exclusion is reached after step 3 and the exclusive session lasts untill step 4. After step 3, *Account* widget shown in Figure 3(a) is used to perform income appending actions. The usage manual is offered to participants to assist them in solving the problem. The time participant spends learning how to properly apply the mechanism is recorded.

Introduction to the programming-by-demonstration assignment

The goal of the programming-by-demonstration assignment is programming of mutual exclusion based on the mechanism that is used in the previous assignment. The introduction is necessary in order to have participants

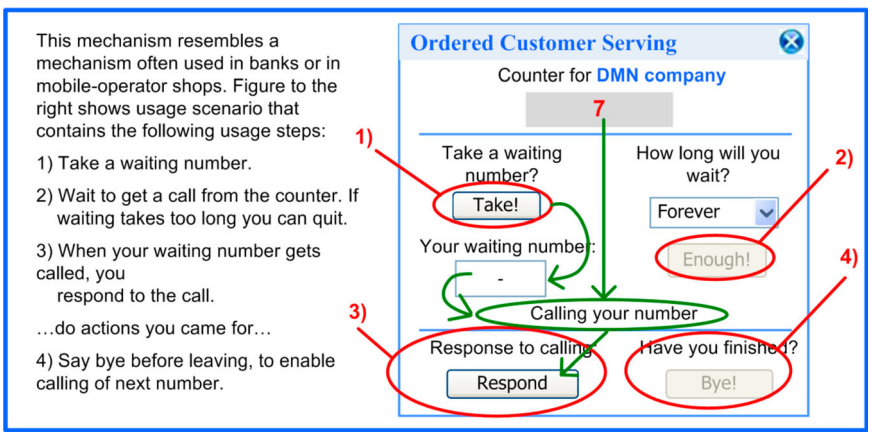

Figure 4. User manual to help participants to achieve mutual exclusion using the mechanism *Ordered Customer Serving*.

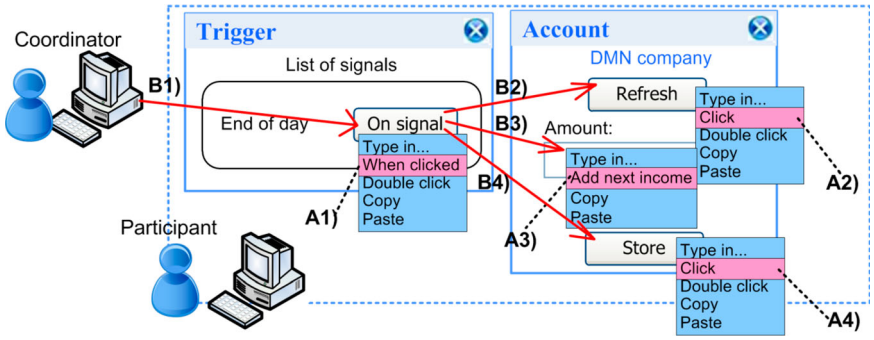

Figure 5. Programming of execution of actions upon reception of a signal with the *Trigger* widget.

prepared for using a special programming tool. The idea behind the programming of mutual exclusion is to use Geppeto – a widget-oriented programming by demonstration tool. The set of GUI actions which were performed manually in the interactive assignment should be defined to execute as a run-time program using Geppeto. As part of the introduction, Geppeto programming paradigm is demonstrated and new elements of the programming environment are explained.

Geppeto paradigm [28,29] supports event-driven programming model well suited for programming over graphical interfaces, like buttons and textboxes. Only a subset of the Geppeto's programming elements is adopted and customized for the purpose of mutual exclusion. Interface to programming is exposed through buttons – central graphical constructs commonly recognized by non-programmers as elements listening to user clicks and triggering designated action upon click. While in traditional Web applications button-assigned actions are fixed and their programming is out of reach of end-user, Geppeto allows end-user to define a sequence of actions which gets assigned to a listening button. In response to a certain type of event, typically a click on the listening button or a specific signal from the run-time environment, the listening button triggers execution of a defined sequence of actions.

Mutual exclusion of the operations of income appending needs to be programmed to execute when the *end of a day* signal occurs. Triggering is performed by using the Geppeto environment and a special purpose widget shown in Figure 5. The *Trigger* widget provides functionality for receiving the *end of a day* signal that is signaled by the coordinator. Participants are tutored on the usage of the *Trigger* widget.

Besides visually representing reception of a signal, the *Trigger* widget allows execution of a sequence of actions upon reception of a signal. Figure 5 demonstrates programming of actions which execute upon reception of a signal. The button labeled *On signal*, which represents an interface for listening to the signal reception, needs to be configured first. Signal occurrence enforces a click on the button. Signal-initiated click on the button results with the same effect as a click performed by a user. Hence, listening for a signal is configured by right-clicking on the listening button and then selecting the action "*When clicked*" from a context menu (A1). A sequence of actions which will be executed upon reception of a signal is programmed next.

The example shows programming of actions which perform income appending operations on the *Account* widget. Each action that is programmed has a targeted GUI element on which it executes. The actions are programmed on the targeted GUI element by selecting one of the listed actions from a drop-down menu which is specific to the element. A click on the *Refresh* button is the first action which is defined (A2). The following is the action of adding next income (A3) which is a special purpose action of the textbox element. The action retrieves participant's daily income from the server and sums it to the value of income present in the textbox at the given time of execution. Execution of this action ends by leaving the summed value written in the textbox. The last action which is programmed is the click on the *Store* button (A4). Having the *Trigger* widget programmed as demonstrated, an occurrence of the *end of a day* signal (B1) will automatically trigger execution of a sequence of programmed actions. In the figure, B2 denotes triggering of action A2, while B3 and B4 denote triggering of actions A3 and A4 respectively.

Each participant is required to program their instance of *Trigger* and *Account* widget as demonstrated in Figure 5. Correct behavior of programmed functionalities is tested upon the occurrence of the *end of a day* signal. All instances of *Trigger* widget, of which each participant's browser locally hosts one instance, listen to the *end of a day* signal. Upon reception of the signal, each instance triggers independent execution of a sequence of actions. Participants need to confirm valid triggering of their sequence that automates income appending operations. The listening button can be reprogrammed any number of times. After all participants confirm validity, the scenario of income appending is played in whole and the result is analyzed. An explanation is provided to clarify why the resulting amount of income on the company's account is not equal to the sum of all incomes. Participants are

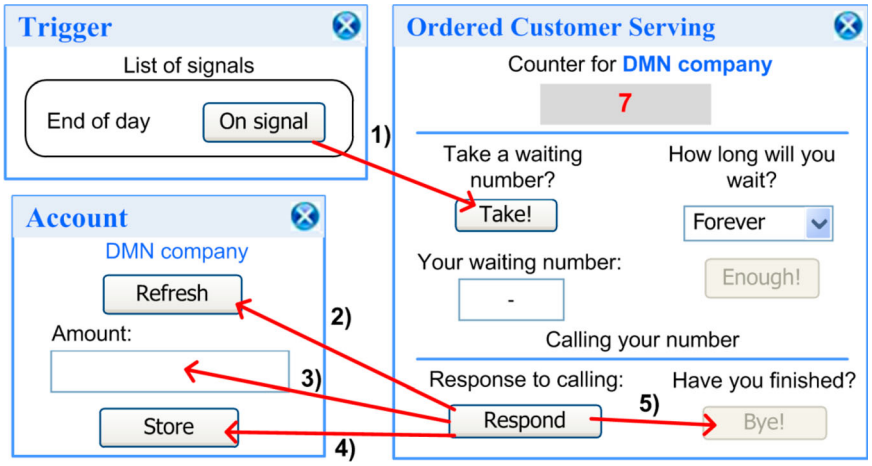

Figure 6. Solution of programming mutual exclusion over graphical interfaces of the mechanism *Ordered Customer Serving*.

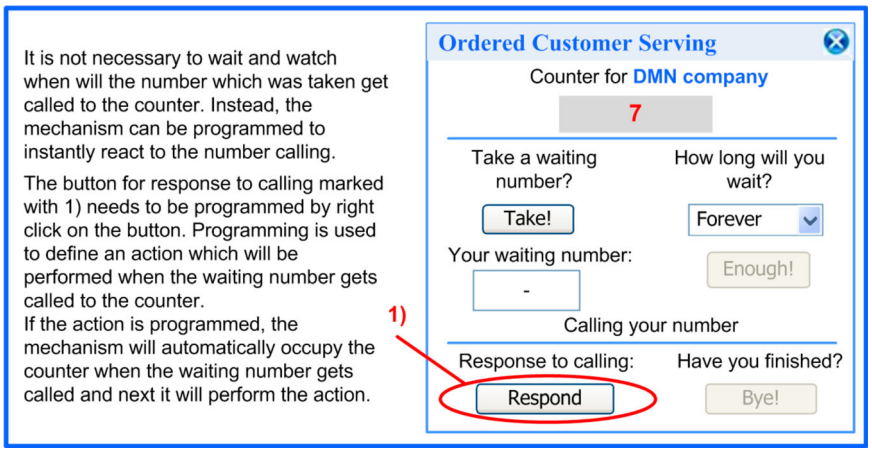

Figure 7. User manual to help programming of mutual exclusion of the mechanism *Ordered Customer Serving*.

reminded of a racing problem, which occurs if multiple sequences of income appending operations execute simultaneously and not in a mutual exclusion.

Programming-by-demonstration assignment

Using the *Ordered Customer Serving* widget and *Geppeto* programming paradigm, participants are required to program the sequence of actions which executes income appending operations in a mutually excluded fashion. The time participant spends learning how to program the mechanism is recorded.

Participants are expected to bind together the logic which is initiated by the *end of a day* signal and the *response to calling* signal which is fired when the number called matches the received *waiting number*. The expected solution is shown in Figure 6.

An action which takes the *waiting number* is programmed to execute upon the occurrence of the *end of a day* signal (1). The *Ordered Customer Serving* widget is programmed to trigger the sequence of income appending operations (2, 3, and 4) to execute automatically when calling out the waiting number occurs. The sequence of income appending operations needs to be followed by the action which ends the exclusive session (5).

In order to achieve a correct solution, usage hints on programming *Ordered Customer Serving* widget are shown to participants in the form of a manual depicted in Figure 7. The key interface element for programming with Geppeto is the button for responding to a calling (1). By right-clicking on the button, participants need to define a sequence which is triggered as an automated response to calling out the waiting number.

Evaluation

The evaluation was performed on two independent groups of participants. Sample group for testing conventional method is referred to as the *Conventional* sample. Sample group for testing experimental method is referred to as the *Experimental* sample. Participants were recruited from students enrolled in mandatory Operating Systems course at the University of Zagreb, Faculty of Electrical Engineering and Computing. Participants have not yet received education on race condition problem and mutual exclusion, but have been educated on writing programs with multiple processes

and multithreaded programs. *The conventional* sample consisted of 11 participants. *The experimental* sample consisted of 14 participants.

The testing procedure for evaluating the conventional method was based on an assignment which requires solving the race condition problem by applying counting semaphore in C programming language (details on the flow of the procedure are depicted in Figure 1). Testing procedure of the experimental method comprised three assignments shown in Figure 2: the interactive assignment, the programming-by-demonstration assignment, and the assignment in C language. The first two assignments utilized a specially designed graphical mechanism and a social game scenario. The third assignment of the experimental method is the same assignment in C language used for evaluating the conventional method.

Results of the conventional method evaluation are presented first, followed by the results of the experimental method evaluation. The results are compared in a discussion which presents authors' reflections and conclusions from the results. Issues which present threats to validity of stated conclusions are presented after the comparison and discussion.

Conventional method

Statistical summary of the results is given in Table 2. Success rate of the conventional method evaluated to 64%, which means 7 out of 11 participants successfully solved the race condition problem by applying counting semaphore in C programming language. Time-on-task analysis considers only the completion times of participants who successfully solved the problem. Time-on-task results are displayed in numbers in Table 2 and in the form of a histogram in Figure 8. Distribution of results on time axis generates a slightly inclined shape of the histogram. Inclined shape points to an unequal distribution of results left and right from the mean value. Such distribution of results causes the shift of the median value from the mean. In this case, median displays a 14 min longer time-on-task than the mean value, which evaluates to 156 min.

Experimental method

Statistical summary of the results is given in Table 3. The table displays measured results for each assignment

Table 2. Metrics showing results of conventional method (mean, median, and standard deviation are displayed in minutes).

	Number of participants	Success rate	Mean	Median	Standard deviation
Conventional sample	11	7/11 = 0.64	156	170	52

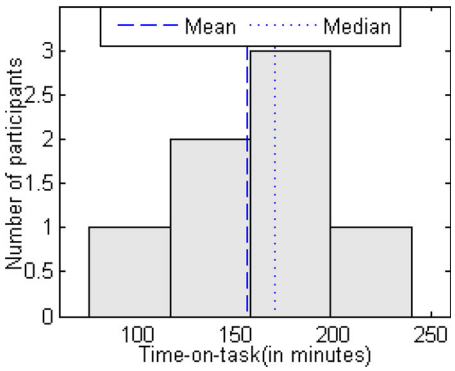

Figure 8. Histogram displaying time-on-task results of conventional method.

of the experimental method and for the overall procedure. The success rate of the overall procedure was 79%, due to the success rate of the assignment in C language, while the first two assignments had 100% success rates. Altogether 11 out of 14 participants successfully solved the race condition problem by applying counting semaphore in C programming language.

Time-on-task analysis considers only the completion times of participants who successfully solved the problem. Time-on-task results are displayed in numbers in Table 3 and in the form of a histogram in Figure 9. The analysis focuses on the cases where the median value and the mean value do not overlap as in normal distribution. The absence of overlapping is noted only in the last assignment of the experimental method, the assignment that involves mutual exclusion by counting semaphore in C programming language. In the interactive assignment, the median and the mean value almost overlap at 4 min. In the programming-by-demonstration assignment, the median and the mean value almost overlap at 11.5 min. In the last assignment, the median value is 150 min, while the mean value is 170 min. In the overall procedure, median value displays a 15 min shorter time-on-task than the mean, which evaluates to 185 min.

Figure 9 shows histograms of time-on-task for each assignment of the experimental method and for the overall procedure. Histograms show that d) has the greatest influence on the overall time-on-task shown in a). The influence of the results shown in d) is based on the variance and the mean time-on-task which are by far greater in d) than in b) and c). The same shape of a slightly inclined histogram can be seen on a) and d) as well as the shift of the mean and the median value which is related to such distribution of results.

Comparison and discussion

The most indicative parameters for comparison of conventional and experimental teaching method are the success rate and time-on-task results of the *Conventional* and the *Experimental* sample.

Table 3. Metrics showing results of experimental method (mean, median, and standard deviation are displayed in minutes).

	Number of participants	Success rate	Mean	Median	Standard deviation
Overall procedure	14	11/14 = 0.79	185	160	71
Interactive assignment	14	14/14 = 1	4.25	4	1
Programming-by-demonstration assignment	14	14/14 = 1	11.25	11.5	2
Assignment in C language	14	11/14 = 0.79	170	150	72

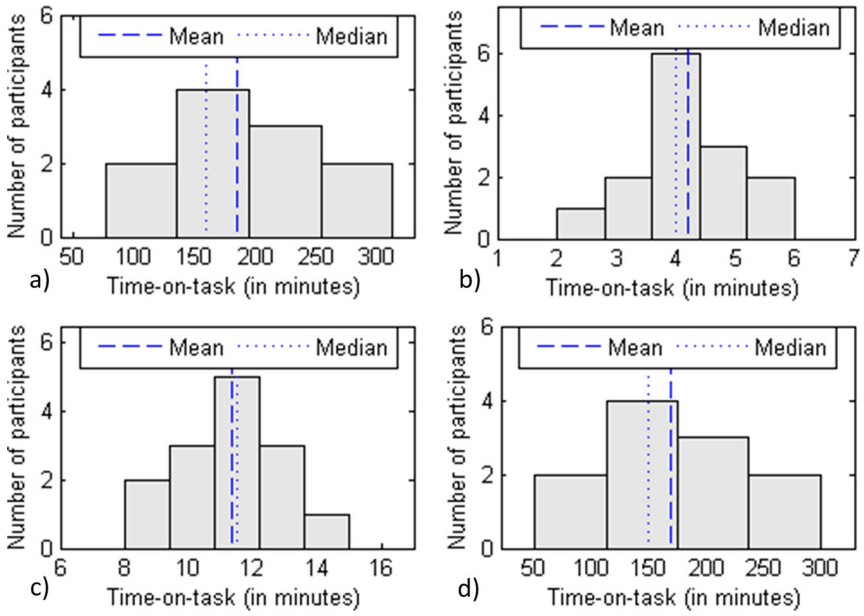

Figure 9. Histograms displaying time-on-task for the following: (a) overall learning process in the experimental method; (b) the inteactive assignment only; (c) the programming-by-demonstration assignment only; d) the assignment in C language only.

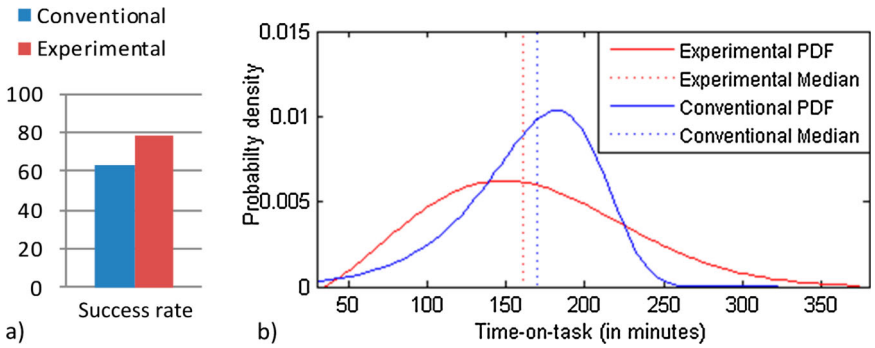

Figure 10. Side-by-side comparison of (a) success rate of the *Conventional* and the *Experimental* sample; (b) estimated probability density functions (PDF) of time-on-task for the *Conventional* and the *Experimental* sample.

Figure 10(a) shows the success rate comparison of the *Experimental* and the *Conventional* sample. The *Experimental* sample has achieved 15% higher success rate in solving the specified race condition problem in C programming language. The explanation of better success rate of the *Experimental* sample is found in the additional teaching steps that are introduced by the experimental teaching method. The introduced steps precede solving the race condition problem in C language and conceivably contribute to better understanding of the technology independent concept of how the solution is reached. Therefore, while programming in the C, participants do not struggle with figuring out the idea behind the solution; instead, they can focus on overcoming the syntax of how the solution is expressed in C language.

In order to analyze whether the additional steps prolonged the overall learning process of the experimental method, the time-on-task measure is examined taking into account all the steps of the learning process. Figure 10(b) shows estimated probability density functions of time-on-task for the *Conventional* and the *Experimental* sample side-by-side. Probability density functions (PDF) are estimated with right- and left-skewed Weibull distributions in order to model the offset of a median and mean values, which is noticeable in the results of both *Conventional* and *Experimental* sample. Since outliers are considered important for the

Table 4. Interquartile range statistics for the estimated probability density functions.

	Lower bound of the interquartile range	Upper bound of the interquartile range
Conventional PDF	139	195
Experimental PDF	119	206

analysis which follows, estimated functions are fit to match histogram data and the measured median values. Mean values of estimated PDF functions are kept in the 95% confidence intervals of the original results. Since the duration of lecturing on the race condition problem is estimated to at least 15 min, the model does not consider times on-task lower than 30 min. Additional 15 min is a fixed minimum amount of time accounted to programming.

Better time-on-task performance of the *Experimental* sample is inferred from the median, interquartile range, and long tail elements of the estimated PDFs. Median values show ten minutes shorter time-on-task in favor of the *Experimental* sample. Interquartile range statistics are shown in Table 4. Lower bound of the estimated interquartile range of the experimental method is 20 min lower than the lower bound of the conventional method. Upper bound of the estimated interquartile range of the experimental method is 11 min higher than the upper bound of the conventional method. The higher upper bound of the experimental method points to the long tail of the estimated distribution, which models the 15% better success rate.

In the measured results of the *Experimental* sample, outliers of the long tail shift the mean so much to the right that it shows higher time-on-task than the mean of the *Conventional* sample. Such shift of the mean convinces the authors in the validness of assigning the 15% better success rate of the experimental method to the long tail in the estimated time-on-task PDF. Therefore, the model assumes 15% of the participants who fail when the conventional method is applied would succeed if the experimental method was applied, but with higher time-on-task. In this sense, the experimental method helps students with minor performance and high determination to reach the solution with extra time invested.

Additional discussion is devoted to the opposite aspect of outliers and how the model estimates time-on-task for the best-performing students. Outliers with particularly low time-on-task represent the best students who have a high mental capacity to quickly understand and absorb new and abstract concepts. In this sense, the best-performing students are delayed by the extra time spent in additional steps of the experimental method. Outliers with particularly low time-on-task are modeled in favor of the conventional method. The estimation of such results is modeled by the long left tail of the probability distribution of conventional method and the shift of the mean value to the left of the median.

Threats to validity

Several issues are outlined which present threats to validity. Authors are aware the results are performed on a small sample size. The obtained results have provided positive feedback on the validity of the experimental method and can serve as an argument to plan a larger experiment in the future. Also, the validation of the experimental method is compared to the conventional teaching method only. The validation with other non-conventional teaching methods is the next important concern that will be addressed in the future work. In addition to non-conventional teaching methods listed in the related work, flipped classroom approach is reported to be successfully applied in numerous teaching fields [31,32] and is considered as a candidate for new experiments in the future work.

The participants have self-reported time-on-task for the conventional method. In its nature, the conventional method is based on an individual assignment that students perform at home. It often results in lengthy times needed for solving the assignment. While self-report is considered less reliable method than the actual measurement of time-on-task by the experimenters, special instructions and additional practice was undertaken with the participants to train the participants to perform self-reporting as correctly as possible.

Two independent sample t-test was evaluated on measured time-on-task results of the *Conventional* and the *Experimental* sample with failure to reject the null hypothesis at the significance level of 0.05. Wilcoxon rank-sum test was also evaluated with failure to reject the null hypothesis at the same significance level. Accordingly, the decision to estimate independent probability density functions of time-on-task for the experimental and conventional method is arguable. In this sense, more simplified conclusions about time-on-task could state that the experimental method is not worse than the conventional method. However, authors find that the models of estimated time-on-task have additional value for interpretation of results. The estimated PDFs were used to randomly generate datasets of the same size as the *Conventional* and the *Experimental* sample. In a million datasets generated from the estimated PDFs, t-test and rank-sum test resulted with failure to reject the null hypothesis in more than 95% of the cases, which is consistent with the results of t-test and rank-sum test on the original data from the *Conventional* and the *Experimental* sample.

Conclusion

The experimental method for teaching mutual exclusion in parallel programming is proposed and evaluated

against the conventional teaching method. The experimental method relies on a specially designed graphical mechanism for teaching mutual exclusion and a social game context which is subtly applied in the teaching process. The game scenario is used to give meaning to the assignments, while other gamification elements are omitted. Two major conclusions are derived from the evaluation results of an experiment comprising 11 participants involved in conventional and 14 participants in experimental method. First, experimental teaching method has 15% higher success rate than the conventional method in solving race condition problem with counting semaphore in C programming language. Second, the experimental method is slightly advantageous when median time-on-task results are compared. The time-on-task comparison is based on median values, due to the impact of outliers on the mean. Outliers could not be discarded from the results since correlation is confirmed between the number of outliers and the higher success rate. The correlation is modeled with the long tail of the estimated time-on-task probability distribution for the experimental method.

Note

1. Operating Systems course at the University of Zagreb, Faculty of Electrical Engineering and Computing, http://www.fer.unizg.hr/en/course/os.

Acknowledgement

The authors acknowledge the support of the Croatian Science Foundation through the Recommender System for Service-oriented Architecture (IP-2014-09-9606) research project. The authors wish to thank all the members of the CCL Laboratory at the Faculty of Electrical Engineering and Computing, assistant professor Siniša Popović, associate professor Marin Golub, associate professor Domagoj Jakobović, assistant professor Stjepan Groš, and assistant professor Ante Đerek for their help with preparing this paper.

Disclosure statement

No potential conflict of interest was reported by the authors.

Funding

The authors acknowledge the support of the Croatian Science Foundation (Hrvatska Zaklada za Znanost) through the Recommender System for Service-oriented Architecture research project [grant number IP-2014-09-9606].

References

[1] NSF/IEEE-TCPP Curriculum Initiative. NSF/IEEE-TCPP Curriculum Initiative on Parallel and Distributed Computing – Core Topics for Undergraduates, Version 1; 2012. [cited 2017 Jun 1]. Available from: https://grid.cs.gsu.edu/ tcpp/curriculum/.

[2] Joint Task Force on Computing Curricula, Association for Computing Machinery (ACM) and IEEE Computer Society. Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science; 2013; ACM; New York (NY), USA.

[3] Sivilotti PAG, Pike SM. A collection of kinesthetic learning activities for a course on distributed computing. ACM SIGACT News (ACM New York, NY, USA); 2007 Jun; 38 (2). Sect. Distributed Computing (col: 26). p. 56–74.

[4] Shene CK, Carr S. The design of a multithreaded programming course and its accompanying software tools. J Comput Small Coll. 1998;14:12–14.

[5] Dijkstra EW. Solution of a problem in concurrent programming control. Commun ACM. 1965;8(9):569. doi:10.1145/365559.365617.

[6] Lamport L. A new solution of Dijkstra's concurrent programming problem. Commun ACM. 1974;17(8):453–455. doi:10.1145/361082.361093.

[7] Hoare CAR. Monitors: an operating system structuring concept. Commun ACM. 1974;17(10):549–557.

[8] Lee EA. The problem with threads. Computer (Long Beach Calif). 2006;39(5):33–42. doi:10.1109/MC.2006.180.

[9] Arroyo M. Teaching parallel and distributed computing to undergraduate computer science students. Proceedings of the 27th International Parallel and Distributed Processing Symposium Workshops & PhD Forum; IEEE Conference Publications; 2013 May 20–24; Cambridge (MA), USA. p. 1297–1303.

[10] Adams J, Brown R, Shoop E. Patterns and exemplars: compelling strategies for teaching parallel and distributed computing to CS undergraduates. Proceedings of the 27th International Symposium on Parallel and Distributed Processing Workshops and PhD Forum; 2013; IEEE Computer Society; Washington (DC), USA. p. 1244–1251.

[11] Von Praun C. Parallel programming: design of an overview class. Proceedings of the 2011 ACM SIGPLAN X10 Workshop, Article No. 2; 2011 Jun 4; San Jose (CA), USA. New York (NY), USA: ACM.

[12] Scott M. Don't start with Dekker's algorithm: Top-down introduction of concurrency. Poster session presented at: Workshop on Directions in Multicore Programming Education; 2009 Mar 8; Washington (DC), USA.

[13] Gregg C, Tychonievich L, Cohoon J, et al. EcoSim: a language and experience teaching parallel programming in elementary school. Proceedings of the 43rd ACM Technical Symposium on Computer Science Education; 2012; ACM; New York (NY), USA. p. 51–56.

[14] Malnati G, Cuva CM, Barberis C. JThreadSpy: teaching multithreading programming by analyzing execution traces. Proceedings of the 2007 ACM workshop on Parallel and distributed systems: testing and debugging; 2007; ACM; New York, NY, USA. pp. 3–13.

[15] Carr S, Mayo J, Shene CK. Threadmentor: a pedagogical tool for multithreaded programming. J. Educ Resour Comput. 2003 Mar;3(1):1–es. Article No. 1.

[16] Manickam V, Aravind A. If a picture is worth a thousand words, what would an animation be worth? Proceedings of the 16th Western Canadian Conference on Computing Education; 2011; ACM; New York (NY), USA. p. 28–32.

[17] Offenwanger A, Lucet Y. ConEE: an exhaustive testing tool to support learning concurrent programming synchronization challenges. Proceedings of the Western Canadian Conference on Computing Education; 2014; ACM; New York (NY), USA. Article No. 11.

[18] Putchala MK, Bryant AR. Synchron-ITS: an interactive tutoring system to teach process synchronization and shared memory concepts in an operating systems course. International Conference on Collaboration Technologies and Systems (CTS); 2016 Oct 31; Orlando (FL), USA. IEEE Conference Publications. p. 180–187.
[19] Domínguez A, de Navarrete JS, de Marcos L, et al. Gamifying learning experiences: practical implications and outcomes. Comput Educ. 2013;63:380–392.
[20] Sung HY, Hwang GJ. A collaborative game-based learning approach to improving students' learning performance in science courses. Comput Educ. 2013;63(1): 43–51.
[21] Attalia Y, Arieli-Attalib M. Gamification in assessment: do points affect test performance? Comput Educ. 2015;83:57–63.
[22] Hanus MD, Fox J. Assessing the effects of gamification in the classroom: a longitudinal study on intrinsic motivation, social comparison, satisfaction, effort, and academic performance. Comput Educ. 2015;80: 152–161.
[23] Wiese J, Das S, Hong JI, et al. Evolving the ecosystem of personal behavioral data. Hum-Comput Interact. 2017:1–64. doi:10.1080/07370024.2017.1295857.
[24] Erhel S, Jamet E. Digital game-based learning: impact of instructions and feedback on motivation and learning effectiveness. Comput Educ. 2013;67:156–167.
[25] Le Hénaff B, Michinov N, Le Bohec O, et al. Social gaming is inSIDE: impact of anonymity and group identity on performance in a team game-based learning environment. Comput Educ. 2015;82:84–95.
[26] Orosz G, Farkas D, Roland-Lévy C. Are competition and extrinsic motivation reliable predictors of academic cheating? Front Psychol. 2013;4(87):1–16.
[27] Christy KR, Fox J. Leaderboards in academic contexts: a test of stereotype threat and social comparison explanations for women's math performance. Comput Educ. 2014;78:66–77.
[28] Srbljić S, Škvorc D, Skrobo D. Widget-oriented consumer programming. Automatika: J. Control, Meas, Electron, Comput Commun. 2009;50(3–4):252–264.
[29] Vladimir K, Budiselić I, Srbljić S. Consumerized and peer-tutored service composition. Expert Syst Appl. 2015;42(3):1028–1038.
[30] Budiselić I, Vladimir K, Srbljić S. Component recommendation for composite application development. Expert Syst Appl. 2015;42(22):8573–8587.
[31] Adams C, Dove A. Calculus students flipped Out: the impact of flipped learning on calculus students' achievement and perceptions of learning. PRIMUS. 2018;28(6):600–615.
[32] Pineda UR, Minchala OE, Auquilla DO. Using the flipped classroom to teach educational models in english at the education national university (UNAE) of Ecuador. Speech, Lang Hear. 2018;21(2):94–97.