




**School of Computing Technologies**  
**COSC1114 Operating Systems Principles**  
**Project 1 – Multithreading & Synchronisation**

	<b>Assessment Type:</b> Group project (maximum of two students) Submit online via <a href="#">Canvas</a> → <a href="#">Assignments</a> → <a href="#">Assignment 1</a> Clarifications/updates may be made via announcements and relevant discussion forums.
	<b>Due Date:</b> Monday, August 26, 2024, at 23:59.
	<b>Weighting:</b> 30 marks that contributes 30% of the total assessment.

### 1. Overview

This assignment focuses on the concept of multithreading. In this assignment, you will implement two tasks to investigate how multithreading enables parallel computing and the challenges that must be managed to ensure consistency.

### 2. Learning outcomes

This assessment is relevant to the Course Learning Outcomes CLOs 2, 5, and 6.

### 3. Assessment details

This assessment will determine your ability to

1. Understand the concepts taught over the first 5 weeks of the course.
2. Work independently in self-directed study to research the identified issues.

It is your responsibility to correctly submit your files. Please verify that your submission is correctly submitted by downloading what you have submitted to see if your submitted file includes the correct content.

Never leave submission to the last minute – you may have difficulty uploading files. **However, if your final submission is after the due time, late penalties will apply.**

### 4. Academic integrity and plagiarism

It is understood by us that many of the algorithms used in this course have common implementations. You are welcome to look at online code examples to understand possible solutions to the set problems. However, what you submit must be your own work and your submission will be checked and compared with other solutions.

**Submitting material generated by an AI tool as your own work constitutes plagiarism and academic dishonesty. DO NOT simply copy other people's work, it is not difficult for us to detect copied work and we will pursue such cases.**

## 5. Group work

**The group work should be completed in pairs, that is, with no more than two students per group (you may form groups with any student in the course).**

You have the option to complete this assignment on your own.

Students are required to contribute to and demonstrate collaborative effort on both tasks. All group members will receive the same grade by default.

To ensure effective management of your group work and to demonstrate ongoing contributions to your project, **you will be required to maintain contribution logs that detail your involvement in both tasks.** These should include records of tasks completed, hours spent, and notable interactions within the group. This document can be reviewed if there are disputes about contributions or for grading adjustment purposes. **Should the group disband, each member must be capable of completing the project on their own.**

If there are any issues that affect your work, such as being sick, you must keep your group informed in a timely manner. Your final grade will be based on the work you and your partner complete throughout the duration of the assignment. We will treat everybody fairly, and account for times when issues arise when marking your group work and contributions. However, you must be upfront and communicate with us. If you fail to inform us of issues in a timely fashion and we deem that your actions significantly harm your group's performance, we may award you a reduced grade. It is academic misconduct if you are deliberately dishonest, lie to, or mislead your group or teaching staff in a way that harms your group.

Notify your tutor immediately if you encounter any issues working with your group at any time. We will do our best to help manage group issues, so that everybody receives a fair grade for their contributions.

## 6. Submission

Your assignment should submit via [Canvas](#) → [Assignments](#) → [Assignment 1](#).

**You will submit two things:**

1. Your C++/C source code in [a zip file](#), which includes
  - the code you have developed, and
  - a README file in which **please specify exactly how to run your code on the provided servers** and **explicitly describe where you have implemented locks and condition variables, including the file names and line numbers.**  
**When working in pairs, please include both student numbers in the zip filename, for example, [s1234567\\_s4567890.zip](#); otherwise, use your own student number only.**
2. Besides the zip file, **organize the source code of tasks in a separate text file [s1234567\\_s4567890.txt](#)** (copy & paste them into a text editor and then save it). **This text file is for Turnitin plagiarism check.**

**Your submission will not be graded if you fail to submit your txt or if your txt does not pass the Turnitin plagiarism check.**

## 7. Late submission policy

A penalty of 10% per day of the total available marks will apply for each day being late. **After 10 days, you will receive zero marks for the assignment.**

If you want to seek an extension of time for assignment submission, you must have a substantial reason for that, such as unexpected circumstances. **Reasons such as, unable to cope with study load, is not substantial.** Also, you must apply for an extension as soon as possible. Last minute extensions cannot be granted unless it attracts special consideration.

Please find out how to apply for special consideration online at <https://www.rmit.edu.au/students/student-essentials/assessment-and-results/special-consideration/eligibility-and-how-to-apply>.

Any student wishing an extension must go through the official procedure for applying for extensions and must apply at least a week before the due date. Do not wait till the submission due date to apply for an extension.

## 8. Assignment tasks

### Task 1 – Multithreaded multiple file copying (5 marks)

In this task, each thread copies one file from a source to a destination. Threads run concurrently, each handling its own file copy independently.

Your program should be executed with the following command:

```
./mmcopier n source_dir destination_dir
```

where  $n$  ( $n \leq 10$ ) is the number of files, which corresponds to the number of threads used, to be copied under `source_dir`, and `destination_dir` is the target directory.

For example, `./mmcopier 3 source_dir destination_dir` will copy `source1.txt`, `source2.txt`, and `source3.txt` under `source_dir` to `destination_dir`, using 3 threads.

**Please find the sample `source_dir.zip` in the assignment page. The same folder/directory will be used for grading.**

### Task 2 – Multithreaded single file copying (25 marks)

Write a multi-threaded program that reads data from one file and writes it to another.

**Note: This is intended to emulate the Producer-Consumer Problem. In a real-world scenario, you would not copy a file in this manner.**

- **You will have a ‘team’ of readers and a ‘team’ of writer threads. (5 marks)**

The reader threads will take turns reading lines from the source file and storing them in a shared queue. The writers likewise will take turns removing lines from the queue and writing them to the target file.

**Note: Reading and writing should occur in parallel.** The objective is to develop a shared queue, and you will need to determine the appropriate conditions, such as a shared queue that can accommodate a maximum of 20 lines.

We provide you with a bash script [generate\\_text.sh](#) to generate a source file (using a dictionary) for testing purposes. **Please find the bash script and the dictionary in the assignment page.**

- **Insert locks (10 marks)**

You will need to insert locks into the appropriate places using the [pthread\\_mutex](#) API to restrict other threads from accessing critical sections, thereby avoiding race conditions and deadlocks.

- **Avoid busy waiting (10 marks)**

You are to avoid busy waiting. Busy waiting is when a thread does nothing but check if some condition is met repeatedly. Instead, when you are waiting for a condition to be met you should use a [sleep\(\)](#) function call or if you are aiming for a high distinction, **a condition variable** (see The API section for details). What this means is that if you use the [sleep\(\)](#) solution you can get half the marks here but only a solution using the [pthread\\_cond](#) API can attract full marks.

**Hints: the readers must wait when the shared queue is full, and the writers must wait when the shared queue is empty.**

Your program will be invoked as:

```
./mscopier n source\_file destination\_file
```

where [n](#) will be an integer between **2 and 10**, [source\\_file](#) is the file to be copied and [destination\\_file](#) is the destination (these maybe be any valid filesystem path).

For example: [./mscopier 10 input output](#) will use 10 readers and 10 writers to copy the file [input](#) to the file [output](#).

## Technical Specifications

- Programming Language: Implement the system in C++ or C.
- Concurrency Constructs: Use POSIX threads (pthreads).

To compile C++/C program with [pthread.h](#) library, **you must put [-lpthread](#) just after the compile command** (for example: [gcc -Wall -Werror -o mscopier mscopier.c -lpthread](#)), this command will tell the compiler to execute program with [pthread.h](#) library.

- Error Handling: Your program must gracefully handle and report errors such as failed memory allocation or file access issues.
- Makefile: Create a [makefile](#) to compile your programs.
  - [make all](#) will build two programs [mmcopier](#) and [mscopier](#), at once.
  - [make clean](#) will get rid of object and executable files.
  - Make sure you compile with the following flags and fix the errors before submitting your assignment: [-Wall -Werror](#)

## Build Environment

Please note that it is expected that your program will compile and run cleanly on the provided servers:

[titan.csit.rmit.edu.au](#)

[jupiter.csit.rmit.edu.au](#)

[saturn.csit.rmit.edu.au](#)

**Your code must execute on the designated servers. A 20% penalty will be applied if the markers are unable to run your program on these servers.**

### Code Quality

Please note that while code quality is not specifically marked for, it is part of what is assessed as this course teaches operating systems principles partly through the software you develop. **As such lack of comments, lack of error checking, good variable naming, avoidance of magic numbers, etc., may be taken into consideration by your marker in assigning marks for the components of this assignment, although operating systems principles do take priority.**

### Memory Correctness

All accesses to memory are expected to be correct (no reading from uninitialized memory, out of bounds access, not freeing memory that was not allocated) and we will check for these things with [valgrind](#) on the provided UNIX servers using the following command:

```
valgrind --track-origins=yes --leak-check=full --show-leak-kinds=all executable_name args
```

Please check your code prior to submission with [valgrind](#) and if you are still unsure how to do this, ask the teaching staff.

**Memory incorrect code will attract a deduction especially as this is an operating systems course. A 10% penalty will be imposed if the program exhibits memory leaks.**

## 9. Rubric and marking guidelines

The rubric can also be found in **Canvas → Assignments → Assignment 1.**

**Please read “IMPORTANT NOTES” carefully.**

For each assessment requirement, the rubric is designed to be general and the details of what you can improve on will be outlined by your marker in the comments. Nevertheless, this should be treated as a guide to what to aim for.

- **Little to no attempt (0%):** Not attempted.
- **Poor (25%):** Some attempt but did not get the code working or does not compile.
- **OK (50%):** Code compiles but threading is not correctly implemented.
- **Good (75%):** Threading works but there are minor problems.
- **Excellent (100%):** Well-done.

**Please note that code with segmentation faults in the respective part of the assignment we are assessing cannot get above 50%, and code with problems highlighted by [valgrind](#) cannot get over 90% in that part of the assignment.**

## 10. The API

**Note:** this is a basic API of the functions we want you to use for concurrency and thread safety. Please refer to the man pages for these functions on [titan.csit.rmit.edu.au](http://titan.csit.rmit.edu.au) for further details. The [man](#) pages also have examples of how to use these functions.

**Please note that all functions return result values which you are expected to check in your code.**

```
int pthread_create(pthread_t *thread, const pthread_attr_t *attr,
    void *(*start_routine)(void*), void *arg);
```

The `pthread_create()` function creates a thread with the specified attributes. The new thread starts execution by invoking `start_routine()`; `arg` is passed as the sole argument of `start_routine()`.

**Hint:** a void pointer is just a pointer that can point to any pointer type. `attr` can be ignored for this assignment (just pass in a `nullptr`, the default attributes shall be used).

Find out more on [titan](#): `$man pthread_create`

Your program should wait for the threading function to complete by calling the function:

```
int pthread_join(pthread_t thread, void **retval);
```

The `pthread_join()` function waits for the thread specified by `thread` to terminate. If that thread has already terminated, then `pthread_join()` returns immediately. The thread specified by `thread` must be joinable.

Find out more on [titan](#): `$man pthread_join`

The following functions are used to manipulate mutexes which are variables that enforce mutual exclusion. A mutex can only be locked by one thread at a time and all other threads that try to lock that mutex will sleep until the mutex becomes available.

```
int pthread_mutex_init(
    pthread_mutex_t *mutex,
    const pthread_mutexattr_t *attr);
```

The `pthread_mutex_init()` function shall initialize the mutex referenced by `mutex` with attributes specified by `attr`.

**Hint:** you can pass in `nullptr` for `attr` and the default attributes are used.

Find out more on [titan](#): `$man pthread_mutex_init`

```
int pthread_mutex_destroy(pthread_mutex_t *mutex);
```

The `pthread_mutex_destroy()` function shall destroy the mutex object referenced by `mutex`; the mutex object becomes, in effect, uninitialized.

Find out more on [titan](#): `$man pthread_mutex_destroy`