

CS330 – Introduction to Operating Systems

COURSE PROJECT: LINUX PROCESSES, SYNCHRONIZATION & PERFORMANCE

Total: 10 Marks | Group Project (3-4 Students per Group)

Project Information	
Course	CS330 – Introduction to Operating Systems
Project weight	10 Marks
Group size	Three or Four (3-4) students per group
Project type	Group-based project. The group must complete all parts (Parts A–D, the Required Analysis, the Report, and the Viva).
Due date	Week 13, 2026 (Thursday Nov 5th)
Submission	Through the LMS only
Re-submission	After a project has been graded, re-submission with the intention of improving the score is not allowed.

1. Project Overview

This project gives students hands-on experience with Linux in a virtual machine, process creation, inter-process synchronization, shared resources, race conditions, and performance evaluation. It emphasizes virtual-machine/Linux installation, fork()-based process creation, and execution-time experiments. Instead of a large matrix-multiplication programming task, the project uses a synchronization task so that students focus directly on a core Operating Systems concept.

This is a **group-based project (group of three or four students)**. Both members must take part in all parts of the project and must be able to explain every part during the Discussion(Section 12).

2. Learning Objectives

- Install and configure Linux inside a virtual machine.
- Use basic Linux terminal commands and system-monitoring tools.
- Create multiple processes using fork().
- Understand race conditions when processes access a common resource.
- Implement synchronization using a Linux inter-process synchronization mechanism such as POSIX named semaphores.
- Compare an unsynchronized version with a synchronized version.
- Measure execution time and explain synchronization overhead.
- Interpret experimental results using tables and graphs.

Marks Distribution by Part

Part	Title	Section	Marks
Part A	Install the Virtual Machine	Section 4	1.0
Part B	Linux Installation and Configuration	Section 5	1.0
Part C	Synchronization Project Task (implementation)	Section 6	4.0

Part	Title	Section	Marks
Part D	Experiments and Measurements	Section 7	1.0
Analysis	Required Analysis and Graphs	Section 8	1.0
Report	Report Structure, Quality and Presentation	Section 9	0.5
Discussion	Demonstration and Understanding	Section 12	1.5
Total			10.0

3. Required Environment

Component	Requirement
Virtual Machine Manager	Oracle VirtualBox (recommended) or VMware Workstation
Linux	Ubuntu 22.04 LTS or Ubuntu 24.04 LTS
VM allocation	Recommended: 2–4 CPU cores, 4 GB RAM, 25 GB disk
Tools	Terminal, GCC, Make (optional), time/top/htop
Submission	One Word report containing evidence, results, analysis, and code appendix

4. Part A – Install the Virtual Machine [1.0 Mark]

1. Install Oracle VirtualBox on your computer. VMware Workstation may be used instead.
2. Create a new virtual machine named CS330-Linux.
3. Select Linux / Ubuntu (64-bit).
4. Allocate approximately 4 GB RAM and 2–4 virtual CPU cores if your host computer permits.
5. Create a virtual hard disk of approximately 25 GB or more.
6. Attach the Ubuntu ISO image to the virtual optical drive.
7. Start the VM and complete the Ubuntu installation.
8. Create a student account and password. Record the username only; do not include passwords in the report.
9. Restart the VM and confirm that Ubuntu boots successfully.
10. Take screenshots showing the VM settings and the running Ubuntu desktop.

Evidence required: at least screenshots of the VM configuration, the Ubuntu installation/running system, and successful terminal access.

Grading (1.0 mark): VM configuration is correct and matches the requirements (0.5); screenshots are clear, complete, and show the required evidence (0.5).

5. Part B – Linux Installation and Configuration [1.0 Mark]

1. Open Terminal.
2. Update the package list using the normal Ubuntu package-management procedure.
3. Install GCC/build tools if they are not already installed.
4. Verify the compiler installation.

5. Create a project directory, for example: CS330_Synchronization_Project.
6. Inside the directory create folders such as src, results, screenshots, and report.
7. Verify basic commands such as pwd, ls, cd, mkdir, cp, mv, rm, ps, and top/htop.
8. Record the Linux version, CPU information, number of visible cores, and RAM for the report.

Do not paste passwords, private keys, or personal account information into the report.

Grading (1.0 mark): Ubuntu works and the compiler is installed and verified (0.5); project directory, command verification, and system information are recorded correctly with screenshots (0.5).

6. Part C – Synchronization Project Task [4.0 Marks]

Scenario: Shared Counter / Shared Resource

Several worker processes represent concurrent users accessing the same shared resource. Each process must perform a fixed number of updates to a shared counter. The group will demonstrate what can happen when concurrent processes access a shared resource without synchronization, and then protect the critical section using a semaphore.

6.1 Key Terms and Definitions

The following terms are used throughout the project. Use them consistently in your code, tables, and report.

Term	Definition
Update	One execution of the critical section by a process: read the shared counter, add 1, and write the result back (for example, counter = counter + 1).
Updates/Process	The number of updates that each child process performs (the loop count of each worker). This project uses processes created with fork() , not threads, so the unit is “updates per process”.
Processes	The number of child (worker) processes created by the parent with fork(). The parent only creates the children, waits for them, and reports the result; it does not update the counter.
Expected total	Processes × Updates/Process. This is the value the shared counter must hold at the end if no update is lost.
Final counter	The value of the shared counter read by the parent after all children have finished.
Lost updates	Expected total – Final counter. It is zero when update is lost.
T_no_sync	The average wall-clock execution time, in seconds , of the version WITHOUT synchronization (average of at least three runs) for a given configuration.
T_sync	The average wall-clock execution time, in seconds , of the version WITH semaphore synchronization (average of at least three runs) for the same configuration.
Synchronization overhead	The extra time caused by synchronization, expressed as a percentage of the unsynchronized time: $((T_{\text{sync}} - T_{\text{no_sync}}) / T_{\text{no_sync}}) \times 100\%$.
Critical section	The part of the code that accesses the shared counter and must be executed by only one process at a time.
Run	One complete execution of the program, starting with the counter set to zero and ending after the parent has reported the final counter.

6.2 Required Implementation

1. Create a shared integer counter in shared memory (for example, using `mmap()` with `MAP_SHARED` | `MAP_ANONYMOUS`, or `shm_open()`).
2. Create multiple child processes using `fork()`.
3. Each child performs the same number of counter updates.
4. Run **Version A WITHOUT synchronization**. The final counter may differ from the expected value because multiple processes can interfere while updating the shared resource.
5. Run **Version B WITH synchronization**. Use a POSIX named semaphore (or another instructor-approved Linux inter-process synchronization mechanism) to protect the critical section.
6. Ensure the semaphore is opened/created before workers access it and is properly closed and removed after execution (`sem_open`, `sem_close`, `sem_unlink`).
7. Use `wait/waitpid` so the parent process waits for all children before reporting the final result.
8. Measure execution time for both versions (see Section 7.1 for the required timing method).
9. Repeat the experiment with different numbers of processes and different workloads.
10. Explain why the synchronized version is correct and why synchronization can introduce overhead.

6.3 Suggested Experimental Values

Experiment	Processes	Updates/Process	Expected total updates
E1	1	100000	100000
E2	2	100000	200000
E3	4	100000	400000
E4	6	100000	600000
E5	8	100000	800000
E6	4	250000	1000000
E7	4	500000	2000000

The group may adjust the workload if execution is too short or too long. The same experimental settings must be used when comparing the synchronized and unsynchronized versions.

6.4 Implementation Notes

- Declare the shared counter as volatile (or otherwise prevent the compiler from keeping it in a register) so that the race condition can actually occur; compile both versions with the same optimization level and state it in the report.
- Compile the synchronized version with `-pthread` (POSIX semaphores), for example: `gcc -O0 -Wall -pthread sync.c -o sync`.
- The counter must be reset to zero, and the semaphore re-created (initial value 1), at the start of every run.
- Only the critical section (the counter update) should be inside `sem_wait()/sem_post()`. Do not place the whole loop inside the critical section.
- With a single process (E1) no race can occur; the group must explain why the unsynchronized result is still correct there.
- The race condition is non-deterministic. If the unsynchronized version happens to give the correct value in a run, report it honestly and explain; do not force or fake results.

6.5 Grading of Part C (4.0 Marks)

#	Component	What is assessed	Marks
1	Process creation	Correct use of fork(), one shared counter, parent/child coordination with waitpid(), no zombie processes	1.0
2	Unsynchronized shared-resource experiment (Version A)	Race condition demonstrated with evidence (final counter $\neq$ expected) and explained	1.0
3	Synchronization implementation (Version B)	Correct semaphore-based protection of the critical section; correct open/close/unlink cleanup	1.0
4	Correctness and verification	Expected vs. actual results clearly verified for both versions	1.0
Total for Part C			4.0

7. Part D – Experiments and Measurements [1.0 Mark]

For each configuration, run the program **at least three times** and report the average execution time. Keep the machine configuration unchanged during the experiments. All seven experiments (E1–E7) must be completed for both versions.

7.1 Timing Method

- **Metric:** wall-clock (elapsed) time, in seconds, reported with at least three decimal places.
- **How to measure:** inside the parent program, call `clock_gettime(CLOCK_MONOTONIC, ...)` immediately before the first `fork()` and again immediately after the last `waitpid()` returns; the difference is the execution time of the run.
- **What is included:** process creation, the update loops, waiting, and (for Version B) all semaphore waiting. Semaphore creation and removal (`sem_open/sem_unlink`) are outside the timed region. State this clearly in the report.
- **Cross-check:** run at least one configuration with the Linux `time` command and compare the “real” value with your in-program measurement.
- The same timing method must be used for both versions and all experiments.

7.2 Run Protocol

1. Use the same VM settings (CPU cores, RAM) for all experiments. Do not change them between runs.
2. Close unnecessary applications in the VM and the host. If using a laptop, connect it to power.
3. Perform one warm-up run per configuration and discard it.
4. Perform at least three recorded runs for every configuration and version. Alternate between versions to reduce bias from background activity.
5. Start every run from a clean state: counter = 0 and a freshly created semaphore.
6. If one run differs from the others by more than 20%, repeat that configuration and mention the outlier in the discussion.
7. Keep the raw terminal output (or a log/CSV file) of the runs; include screenshots of at least one run of each version in the report.

7.3 Data Tables

Table D-1 – Execution time (seconds).

Exp	Proc.	Updates/ Proc.	Version	Run 1 (s)	Run 2 (s)	Run 3 (s)	Avg (s)
E1	1	100000	Without synchronization				
E1	1	100000	With synchronization				
E2	2	100000	Without synchronization				
E2	2	100000	With synchronization				
E3	4	100000	Without synchronization				
E3	4	100000	With synchronization				
E4	6	100000	Without synchronization				
E4	6	100000	With synchronization				
E5	8	100000	Without synchronization				
E5	8	100000	With synchronization				
E6	4	250000	Without synchronization				
E6	4	250000	With synchronization				
E7	4	500000	Without synchronization				
E7	4	500000	With synchronization				

Table D-2 – Correctness (final counter values).

Exp	Expected total	Final counter (no sync) Run 1	Final counter (no sync) Run 2	Final counter (no sync) Run 3	Avg lost updates (no sync)	Sync: correct in all runs? (Yes/No)
E1	100000					
E2	200000					
E3	400000					
E4	600000					
E5	800000					
E6	1000000					
E7	2000000					

Table D-3 – Summary and synchronization overhead.

Exp	Proc.	Updates/ Proc.	T_no_sync (s)	T_sync (s)	Overhead (%)	Lost updates (% of expected)
E1	1	100000				
E2	2	100000				
E3	4	100000				

Exp	Proc.	Updates/ Proc.	T_no_sync (s)	T_sync (s)	Overhead (%)	Lost updates (% of expected)
E4	6	100000				
E5	8	100000				
E6	4	250000				
E7	4	500000				

8. Required Analysis [1.0 Mark]

8.1 Formulas

Quantity	Formula	Unit
Expected final counter	Processes × Updates/Process	updates
Lost updates	Expected total – Final counter	updates
Lost updates (%)	(Lost updates / Expected total) × 100%	%
Average execution time	(Run 1 + Run 2 + Run 3) / 3	seconds
Synchronization overhead	$((T_{\text{sync}} - T_{\text{no_sync}}) / T_{\text{no_sync}}) \times 100\%$	%
Cost per update (optional)	$(T_{\text{sync}} - T_{\text{no_sync}}) / \text{Expected total}$	µs per update

Here $T_{\text{no_sync}}$ and T_{sync} are the average execution times (in seconds) defined in Section 6.1. Show one fully worked example calculation in the report.

8.2 Required Analysis Tasks

1. **Expected value:** calculate the expected final counter for every experiment (number of processes × updates per process).
2. **Unsynchronized version:** compare the final counter with the expected value; report the lost updates and the percentage lost. Explain how the lost updates arise (interleaving of read–modify–write).
3. **Synchronized version:** verify that the final value equals the expected value in every run of every experiment.
4. **Execution times:** compare the average execution times of the two versions for each configuration.
5. **Overhead:** calculate the synchronization overhead using $((T_{\text{sync}} - T_{\text{no_sync}}) / T_{\text{no_sync}}) \times 100\%$ for every experiment.
6. **Why synchronization is necessary:** discuss why it is required even though it increases execution time (correctness versus speed).
7. **System-level factors:** discuss process creation, context switching, waiting, and contention for the critical section, and relate them to your measurements.
8. **Contention:** identify the configuration where contention becomes noticeable and justify the choice with data (see Section 8.3).
9. **Workload effect:** compare E3, E6, and E7 (same number of processes, different workload) and explain how the workload affects time and overhead.
10. **Scalability:** discuss how execution time and overhead change as the number of processes grows beyond the number of virtual CPU cores.

8.3 Identifying Contention

Contention is noticeable when processes frequently find the semaphore already held and must wait. In the report, identify the smallest number of processes at which this becomes visible, using evidence such as: (a) the overhead percentage rises sharply compared with the previous configuration, (b) T_{sync} grows faster than the total number of updates, or (c) T_{sync} increases while $T_{\text{no_sync}}$ stays almost flat. Also state how the number of virtual cores in your VM affects the result.

8.4 Required Graphs

Graph	Content	Requirement
G1 (required)	Number of processes vs. execution time (E1–E5), with one line for each version	Both versions on the same graph
G2 (required)	Number of processes vs. synchronization overhead (%) (E1–E5)	Overhead calculated from Table D-3
G3 (optional)	Workload (updates/process) vs. execution time (E3, E6, E7)	Shows the effect of workload
G4 (optional)	Number of processes vs. lost updates (unsynchronized version)	Shows the race condition

Every graph must have a title, labelled axes with units, a legend (when more than one series is shown), and a numbered caption, and must be discussed in the text. A minimum of two graphs (G1 and G2) is required.

8.5 Discussion Requirements

The Discussion section of the report (approximately 400–600 words) must interpret the results rather than only describe them. It should answer at least the following questions: Why did (or did not) the unsynchronized version fail for each configuration? Why is the semaphore version always correct? Where does the overhead come from (system calls, blocking, context switches)? Why does overhead change with the number of processes and with the workload? What would happen with more processes than CPU cores?

9. Required Report Structure [0.5 Mark]

The report is one Word document. It is the primary evidence for all parts; the marks for the technical content are awarded under Parts A–D and the Analysis, while the 0.5 mark of this section is for organization and presentation.

9.1 Report Sections

Section	Required content	Suggested length
Cover page	Course, project title, names and IDs of both students, group number, section, instructor, semester, and submission date	1 page
1. Introduction and objectives	Purpose of the project, problem statement, objectives, and a short outline of the report	½ page
2. Virtual machine and Linux installation	VM settings, Ubuntu version, installation steps, screenshots with captions, compiler verification, and system specifications (Parts A and B)	2–3 pages

Section	Required content	Suggested length
3. Synchronization concept	Definitions in your own words: shared resource, race condition, critical section, mutual exclusion, and the chosen mechanism (POSIX named semaphore: sem_open, sem_wait, sem_post, sem_close, sem_unlink)	1–2 pages
4. Implementation design	Process structure (parent/children), how shared memory is created, the synchronization sequence, and a flowchart or process diagram (recommended)	1–2 pages
5. Experimental setup	CPU model, number of cores, RAM, Linux version, compiler version and flags, processes, workloads, repetitions, and the timing method	1 page
6. Results	Tables D-1, D-2 and D-3, graphs G1 and G2 (and optional G3, G4), terminal screenshots, and one worked calculation	2–3 pages
7. Discussion	Correctness, performance, synchronization overhead, contention, context switching, and scalability (Section 8.5)	1–2 pages
8. Conclusion	Main findings, lessons learned, limitations of the experiment, and possible improvements	½ page
Appendix A – Source code	Complete, commented source code of Version A and Version B, plus the compile commands (monospaced font)	as needed
Appendix B – Terminal output	Selected terminal output and screenshots not included in the main text	as needed
Appendix C – Contribution statement	Table of what each member did in each part (Section 9.3)	½ page
Appendix D – AI usage declaration	Declaration table required by Section 13 (write “No AI tools were used” if none)	½ page

9.2 Formatting Requirements

- A4 page, 11–12 pt readable font, single or 1.15 line spacing, page numbers on all pages.
- Every table and figure must be numbered, captioned, and referred to in the text.
- Screenshots must be legible and must show the terminal/desktop context (for example, the user name and hostname). Never show passwords.
- Source code must be complete, indented, commented, and readable (not as a screenshot).
- Use your own words. Text copied from the project description, textbooks, or websites without citation is not acceptable.

9.3 Group Contribution Statement

Because this is a group project, Appendix C must state who did what. Both members are expected to contribute equally; individual marks may be adjusted after the viva if the contribution is clearly unequal.

Part / Task	Student 1 (name, ID)	Student 2 (name, ID)
Parts A and B (VM and Linux)		
Part C (coding and testing)		
Part D (experiments)		
Analysis, graphs, and discussion		

Part / Task	Student 1 (name, ID)	Student 2 (name, ID)
Report writing and formatting		

10. Deliverables and Submission

Group work: This is a group-based project (group of two students) that requires the group to complete all of the parts above (Parts A–D, the Analysis, the Report, and the Viva).

10.1 What to Submit

- ONE Word document (.docx) containing everything listed below; the name should follow the pattern CS330_Project_Group##_ID1_ID2.docx.
- Screenshots proving VM and Linux installation.
- System specifications.
- Unsynchronized and synchronized implementations (complete source code in the appendix).
- Experimental results and calculations.
- Tables and at least two graphs.
- Performance discussion and conclusion.
- Contribution statement and AI usage declaration.

10.2 Submission Rules

- **Due date:** Week 13, Thursday 5 November 2026 (exact date and time as posted on the LMS).
- **Submission:** All submissions are through the LMS. Upload the ONE deliverable to the LMS. One member of the group uploads the file; both names must appear on the cover page.
- **Late submission:** Handled according to the course policy announced by the instructor.
- **No re-submission:** After an assignment/project has been graded, re-submission with the intention of improving the assignment score will not be allowed.

11. Rubric – Total 10 Marks

Part	Criterion	Excellent (full marks)	Good / Partial	Marks
Part A	1. VM installation & evidence	Correct VM configuration with clear screenshots	Minor missing or unclear evidence	1.0
Part B	2. Linux installation & configuration	Ubuntu works; tools and system information verified	Minor configuration or evidence issues	1.0
Part C	3. Process creation	Correct use of fork() and parent/child coordination	Some issues but basic process structure works	1.0
Part C	4. Unsynchronized shared-resource experiment	Race-condition behavior is demonstrated and explained	Experiment works but explanation/evidence is incomplete	1.0
Part C	5. Synchronization implementation	Correct semaphore-based protection of the critical section; correct cleanup	Synchronization works with minor issues	1.0
Part C	6. Correctness & verification	Expected vs. actual results clearly verified for both versions	Verification incomplete	1.0

Part	Criterion	Excellent (full marks)	Good / Partial	Marks
Part D	7. Experimental design & data collection	Multiple process counts/workloads, repeated runs, averages, complete tables	Limited experiments or inconsistent methodology	1.0
Analysis	8. Performance analysis	Correct overhead calculations, graphs, and meaningful discussion	Basic analysis with limited interpretation	1.0
Report	9. Report quality & presentation	Clear organization, screenshots, tables, conclusion, readable appendix	Minor formatting/organization problems	0.5
Discussion	10. Demonstration & understanding	Both students can explain synchronization and modify/test the project when asked	Partial understanding	1.5
	Total			10.0

12. Instructor Verification / Discussion [1.5 Mark]

The instructor may ask students to demonstrate the project, explain the synchronization mechanism, explain the experimental results, or make a small modification during the demonstration. Students should therefore understand their own implementation rather than relying only on the written report.

- Both members of the group must attend; each member may be questioned individually on any part of the project.
- Typical tasks: run Version A and Version B live, change the number of processes or the workload, explain each line of the critical section, remove or move `sem_wait()/sem_post()` and predict the result, or explain a value in the results table.
- The viva mark (0.5) is awarded individually: 0.25 for explaining the concepts and implementation and 0.25 for successfully modifying/testing the project. If a student cannot explain the submitted work, the marks of the related parts may also be reduced.

13. Academic Integrity and AI Use

All submitted work must follow PSU academic-integrity requirements. Students must understand and be able to explain their own work, and submissions must be original. Where generative AI is used, students must follow the applicable PSU Generative AI Usage Policy, together with the project-specific limits below.

13.1 Maximum Allowed AI Usage

Maximum AI-assisted content: 20% of the written report

AI-assisted content means text that was drafted, or substantially rewritten, by a generative-AI tool. The 20% limit is measured against the written text of the main report (Sections 1–8), excluding the cover page, tables of raw data, terminal output, and the source code appendix.

For **source code**, AI must **not** be used to generate the core implementation of Part C (shared memory, `fork()`, semaphore logic, timing). Code that is not written by the students themselves counts as a violation.

13.2 Disclosure and Consequences

- The group must include the AI Usage Declaration (Appendix D) listing every AI tool used, the purpose, the affected section, and the estimated percentage. If no AI tool was used, state this explicitly.
- Use of AI above the 20% limit, use of AI for any “not permitted” activity, or failure to declare AI use will be treated as an academic-integrity violation and handled according to PSU regulations, in addition to possible mark deductions.
- The instructor may verify authorship during the discussion, and may use similarity or AI-detection tools as supporting evidence.

AI Usage Declaration (template for Appendix D):

AI tool	Purpose	Section affected	What was used (prompt / output)	Estimated %

Declaration: We confirm that this work is our own, that the information above is accurate, and that we can explain all parts of the project.

Student 1 signature: _____

Student 2 signature: _____