Journal of Education and Educational Research
ISSN: 2957-9465 | Vol. 18, No. 2, 2026

Exploration of Curriculum Construction for Operating System Principles Oriented toward Computational Thinking in the Context of Emerging Engineering Education

Chengqiong Ye *, Shuping Yuan

Anhui Xinhua University, Hefei, Anhui, 230088, China
*** Corresponding author:** Chengqiong Ye

Abstract: The development of Emerging Engineering Education (New Engineering Disciplines) has placed new and higher demands on the cultivation of computer-related professionals, specifically requiring the enhancement of students' computational thinking, systems thinking, and engineering practice abilities. As a core course in computer science, Operating System Principles serves as a crucial bridge connecting computer hardware and software, and is also a naturally powerful vehicle for cultivating computational thinking. Therefore, starting from the context of Emerging Engineering Education, this paper systematically analyzes the various problems faced by traditional Operating System Principles courses in fostering computational thinking. Subsequently, from five dimensions—curriculum objective reconstruction, teaching content optimization, teaching method innovation, practice system construction, and assessment reform—it rigorously explores the path for constructing an Operating System Principles course oriented toward computational thinking. It further integrates mainstream operating system practices to implement computational thinking training. Finally, concrete teaching practices verify the effectiveness of the reforms, thereby naturally providing a replicable approach for similar curriculum reforms.

Keywords: Emerging Engineering Education; Computational Thinking; Operating System Principles.

1. Introduction

(1) Context and Talent Demands of Emerging Engineering Education

With the accelerating advancement of the new technological revolution and industrial transformation, emerging technologies represented by artificial intelligence, big data, cloud computing, the Internet of Things, and blockchain are deeply integrating into various sectors of the national economy. While promoting industrial transformation and upgrading, they also pose new and higher demands on the knowledge structure, competency, and innovative thinking of engineering and technology talents. In 2017, the Ministry of Education officially launched the Emerging Engineering Education research and practice project, clearly stating that its construction should be "guided by fostering virtue, centered on responding to change and shaping the future as the construction philosophy, and taking inheritance and innovation, crossing and integration, coordination and sharing as the main pathways to cultivate diversified and innovative outstanding engineering talents" [1].

As a core field of Emerging Engineering Education, the talent cultivation objectives for computer-related majors have clearly shifted from traditional "knowledge transmission" to "competency cultivation" and "innovation orientation." Therefore, students are required not only to master professional theories and core technical skills but also to possess computational thinking, systems thinking, engineering practice abilities, and interdisciplinary integration capabilities. More importantly, computational thinking, as the most fundamental and typical way of thinking in computer science, is a set of comprehensive mental activities that use basic concepts of computer science to solve problems, design systems, and analyze human behavior. Its essence includes abstraction, decomposition, modeling, algorithm design, automation, and evaluation. Hence, it is a core competency for Emerging Engineering talents and the most natural and powerful bridge connecting theoretical knowledge and engineering practice.

(2) Core Position of Operating System Principles and Its Value for Cultivating Computational Thinking

Operating System Principles is a very typical and important compulsory course for majors such as Computer Science and Technology, Software Engineering, and Artificial Intelligence. It is highly theoretical, practical, and comprehensive. Thus, it naturally connects prerequisite courses like Data Structures and Computer Organization Principles and lays the foundation for subsequent courses such as Compiler Principles and Embedded Systems. It is also the most logical and powerful bridge between computer hardware and software, significantly impacting the learning of follow-up professional courses and the formation of engineering abilities.

The course primarily covers process management, memory management, file systems, and I/O management. Each module involves abstract concepts, complex mechanisms, and rigorous logic, thus inherently containing numerous elements of computational thinking: process scheduling algorithms serve as excellent demonstrations of algorithm design and optimization thinking; virtual memory mechanisms exemplify hierarchical abstraction and spatial mapping thinking; synchronization and mutual exclusion mechanisms naturally illustrate concurrent control and resource management thinking. Therefore, the Operating System Principles course is the most ideal and direct vehicle for cultivating students' computational thinking and is bound to be a key factor affecting their system-level problem-

solving and engineering innovation capabilities.

However, traditional teaching of Operating System Principles suffers from prominent drawbacks such as: emphasis on theory, neglect of thinking cultivation, focus on knowledge memorization, disregard for practical application, and reliance on textbooks, lack of integration with cutting-edge developments. Consequently, computational thinking cultivation is not effectively implemented, failing to meet the requirements for cultivating innovative engineering talents in the context of Emerging Engineering Education. Therefore, this paper, starting from the perspective of Emerging Engineering Education and aiming at computational thinking cultivation as the fundamental goal, systematically explores the construction path for the Operating System Principles course, combines mainstream operating system practices to solve traditional teaching problems, and effectively improves the quality of course education.

2. Problems in Traditional Teaching of Operating System Principles

(1) Lack of Explicit Focus on Computational Thinking in Course Objectives

The objective of traditional Operating System Principles courses is often clearly stated as "to enable students to master the basic concepts, core principles, and classic algorithms of operating systems," which is naturally knowledge-transmission oriented. As a result, computational thinking cultivation is not explicitly and systematically incorporated into the core objectives of the course. During classroom teaching, teachers usually explain algorithms and verify theories starting from core concepts, but rarely actively and systematically guide students through computational thinking methods such as abstraction, decomposition, modeling, and automation. This leads students to "know what but not why," forcing them to memorize knowledge points without being able to naturally and fluently transfer the learned knowledge to solve complex engineering problems. Thus, it is difficult to achieve the genuine educational goal of "promoting thinking and application through learning" [2].

(2) Fixed Teaching Content Out of Sync with Emerging Engineering Needs

- Outdated content and lack of integration with cutting-edge technologies. Currently, most textbooks for Operating System Principles are based on classic operating systems (Windows, traditional Linux). Course content focuses on traditional operating system mechanisms and algorithms, while introducing very little—or none—of the cutting-edge technologies required for Emerging Engineering Education, such as cloud operating systems, containerization technology, microkernel operating systems, and domestic operating systems (openEuler, UnionTech UOS). Thus, the course content cannot meet the industry's demand for new system talents.
- Fragmented knowledge lacking systematic integration. Traditional courses teach modules like process management, memory management, and file systems independently, without establishing a complete knowledge system of "principles—mechanisms—applications—thinking." Consequently, students cannot naturally understand the logical relationships between modules, form genuine systems thinking, or grasp the overall operating mechanism and design philosophy of an operating system from a holistic perspective.
- Theory-practice disconnect and shallow practical content. Traditional teaching often separates theoretical instruction from practical components. Practical sessions usually involve simple exercises like algorithm simulation or concept verification, lacking genuine operating system practice training. Therefore, students cannot naturally integrate theory with actual system operation mechanisms. More seriously, the practical content is not designed around computational thinking, so its cultivation remains superficial.

(3) Monotonous Teaching Methods Weakening Student-centered Learning

Traditional teaching of Operating System Principles commonly employs a single model of "teacher lecture + blackboard/PPT demonstration," which can be categorized as typical "instillation teaching." The teacher is the center of instruction, while students passively receive knowledge, lacking opportunities for active thinking, independent inquiry, and hands-on practice. As a result, highly abstract concepts like process synchronization, virtual memory, and deadlocks are extremely difficult for students to truly master through theoretical explanation alone. This often leads to decreased learning interest, rigid thinking, and inability to naturally activate students' initiative in computational thinking, let alone cultivate their autonomous learning ability and problem-solving skills.

(4) Weak Practical System Leading to Insufficient Computational Thinking Training

Practical teaching is the most natural and effective vehicle for cultivating computational thinking. However, the practical system of traditional Operating System Principles courses has two clearly identifiable major flaws: First, practical session design lacks hierarchy and progression. Basic verification experiments take up a large proportion, while innovative experiments are rare. Thus, it cannot effectively facilitate the gradual improvement of students' thinking abilities. Second, existing experimental platforms are often simulated environments disconnected from real operating system application contexts, failing to fully utilize the practical value of mainstream operating systems. Therefore, students find it difficult to directly master key skills such as system-level operations, debugging, and performance optimization through practice, and practical application scenarios for computational thinking are consistently lacking.

(5) One-sided Assessment Making It Difficult to Measure Computational Thinking Ability

Traditional assessment methods for Operating System Principles are very singular, primarily based on "regular assignments + final written exam." The assessment content mainly focuses on students' memorization and simple application of concepts, principles, and algorithms. Thus, it fails to adequately evaluate students' abilities in computational thinking, system design, and engineering practice. More seriously, current evaluation methods lack process-oriented and comprehensive assessment, paying insufficient attention to students' thinking performance, practical processes, and innovation capabilities during the learning process. Consequently, they cannot naturally stimulate students' initiative to cultivate computational thinking, nor achieve the genuine purpose of "promoting learning and thinking through assessment" [3].

3. Concept and Objectives of Course Construction for Operating System Principles Oriented toward Computational Thinking

Guided fundamentally by the requirements of Emerging Engineering Education, the course adheres to the construction philosophy of "student-centered, computational thinking as the core, engineering practice as support, and industry-education integration as the path," integrating computational thinking cultivation throughout all stages and aspects of the teaching process.

Based on the talent cultivation requirements of Emerging Engineering Education and the core of computational thinking cultivation, this paper reconstructs the objectives of the Operating System Principles course across three dimensions: knowledge, ability, and literacy. It clarifies the core requirements for each dimension to ensure the effective implementation of computational thinking cultivation, as detailed in Table 1.

Table 1. Teaching Objectives and Requirements

Objective Type	Core Requirements
Knowledge Objectives	Master core concepts, principles, and classic algorithms of operating systems; understand cutting-edge technologies such as cloud computing, containerization, and domestic operating systems; build a complete operating system knowledge framework; become familiar with basic applications and operation methods of mainstream operating systems.
Ability Objectives	Apply computational thinking to solve complex engineering problems related to operating systems; master system-level operation, debugging, and optimization skills; possess interdisciplinary integration capabilities and basic system innovation design skills; flexibly apply theoretical knowledge in practical scenarios.
Literacy Objectives	Cultivate systems thinking, engineering ethics, and an innovative spirit; establish ideals of serving the nation through technology and contributing to industry; develop a rigorous and pragmatic academic attitude and teamwork awareness; meet the core literacy requirements of Emerging Engineering talents.

4. Path for Constructing an Operating System Principles Course Oriented toward Computational Thinking

(1) Reconstruct Course Content System and Integrate Computational Thinking Elements

Starting from the layered approach of "solid foundation—capability enhancement—cutting-edge expansion," the course content system is restructured to explicitly and reasonably integrate computational thinking elements into each teaching module. It naturally connects cutting-edge technologies and industrial needs, using mainstream operating system practices as the vehicle linking theory and application, thus truly embedding computational thinking cultivation throughout the content design process.

- Focus on Core Principles, Infuse Foundational Computational Thinking: Around the core modules of the operating system, combined with basic operations of mainstream operating systems, students' foundational computational thinking is cultivated. In the process management module, practical cases guide students to decompose concurrency problems. Through the design and analysis of the Process Control Block (PCB), students grasp the application of abstract thinking and methods for decomposing complex concurrency problems into simpler sub-problems. In the memory management module, analogies like "renting houses" or "locker allocation" in daily life are used to explain hierarchical abstraction and spatial mapping thinking. Analyzing and designing page tables and segment tables strengthens students' modeling and logical reasoning abilities.
- Strengthen Practical Application, Deepen Computational Thinking: Focus on cultivating practical operating system operation and basic programming skills to transform theoretical knowledge into practical abilities, deepening students' computational thinking. Students are guided to master basic operations of mainstream operating systems, including system startup, terminal use, and common Linux commands. Through simple script programming (e.g., process control, file read/write operations), students grasp the application of automation thinking, learning to use programming to solve repetitive system tasks, thereby enhancing their problem analysis and solving abilities, and achieving a transition of computational thinking from "understanding" to "application."
- Expand Cutting-edge Technologies, Explore the Boundaries of Computational Thinking: Integrate cutting-edge knowledge such as cloud computing, microkernels, and domestic operating systems. Using real industry cases, analyze their core mechanisms and application scenarios. Guide students to think about the computational thinking logic behind cutting-edge technologies, understand the application methods of domestic OS, cultivate industrial adaptation thinking and the awareness of serving the nation through technology, and stimulate students' innovative enthusiasm [4].

(2) Innovate Teaching Methods to Stimulate Computational Thinking Vitality

Break the single traditional teaching model by integrating diverse methods such as flipped classrooms, case-based teaching, project-driven learning, and tool-enabled instruction. Combined with hands-on training using mainstream operating systems, these methods stimulate students' active thinking, strengthen computational thinking training, and achieve the teaching transformation of "teacher-guided, student-centered" learning.

- Flipped Classroom + Problem-Driven Learning, Guiding Active Thinking: Before class, teachers distribute MOOC videos and preview materials on online teaching platforms, naturally posing preview questions (e.g., "What is the difference between a process and a thread?"), guiding students to preview independently and think actively. During class, using preview questions and teaching difficulties as entry points, teachers organize group discussions and presentations, providing timely guidance. Simple hands-

on exercises validate theories, helping students internalize knowledge and deepen thinking. After class, teachers assign thinking analysis assignments and practical tasks to consolidate learning and subtly cultivate students' autonomous learning and problem-solving abilities.

- Case Teaching + Analogies, Resolving Abstract Difficulties: For highly abstract concepts and mechanisms in Operating System Principles, combine case teaching and analogical explanations to naturally reduce learning difficulty. Use analogies like "traffic lights" for semaphore-based synchronization and mutual exclusion, "renting houses" for virtual memory space allocation, and "library book borrowing system" for file system management, thus making abstract concepts concrete. Furthermore, incorporating real industry cases (e.g., scheduling solutions in large enterprises, application cases of domestic operating systems) illustrates the practical application of algorithms and mechanisms, allowing students to appreciate the engineering value of computational thinking and naturally enhancing knowledge transfer abilities.
- Project-Driven Learning + Group Collaboration, Strengthening Comprehensive Thinking: Based on course content and practical requirements, break down progressive project tasks naturally and logically. Students complete practical operation and development tasks in groups: basic projects are used for foundational practice and principle verification, consolidating basic computational thinking; comprehensive projects focus on system-level tasks, effectively improving students' system design and problem-solving abilities; innovative projects allow students to choose their own topics, making innovative attempts with cutting-edge technologies, thereby naturally stimulating innovative thinking. Furthermore, the group division of labor subtly cultivates students' teamwork and communication skills, truly achieving "exercising thinking through projects, promoting enhancement through collaboration."
- Tool Empowerment + Visualized Teaching, Intuitive Mechanism Presentation: Using tools provided by mainstream operating systems and third-party tools (e.g., `top`, `ps` commands in Linux, system performance analysis tools), various mechanisms like process scheduling, memory mapping, and resource allocation are clearly demonstrated. This helps students understand abstract principles and reduces learning difficulty, naturally transitioning to hands-on practice. Students become proficient in using tools to solve practical problems, effectively cultivating visualization thinking and data analysis capabilities, truly achieving "theory visualization, practice normalization."

(3) Build a Layered Practice System, Implement Computational Thinking Training

Combined with mainstream operating systems, a three-tier progressive practice system of "verification—design—innovation" is constructed to facilitate the gradual improvement of thinking abilities for students at different levels, ensuring the effective implementation of computational thinking training.

- Verification Layer: Solidify Theoretical Foundation, Train Foundational Thinking: Based on the theoretical modules taught, design basic hands-on experiments in a structured and planned manner, naturally and appropriately integrating theoretical teaching with experimental sessions. This consolidates theoretical knowledge and effectively trains foundational computational thinking. Specific experiment contents include installing and configuring mainstream operating systems, managing users and groups, viewing processes, mounting and unmounting devices, and file/directory operations. Consequently, students master basic OS operations, understand the practical application of core principles, and, more importantly, convert abstract theories into concrete operations, subtly cultivating computational thinking elements such as abstraction, decomposition, and verification.
- Design Layer: Strengthen System Capabilities, Deepen Comprehensive Thinking: Naturally and logically combine Linux system applications with moderately difficult design-oriented investigative tasks. Starting from process creation, termination, and troubleshooting steps, systematically guide students to solve process control problems using systems thinking and innovative thinking, thereby effectively cultivating comprehensive abilities. The "Linux Process Management" task serves as a good example: students first investigate common problems in process control within the Linux environment, then design scripts to implement process creation and signal handling schemes, subsequently using Linux commands like `ps` and `kill` to troubleshoot process anomalies and verify results. This naturally strengthens systems thinking, debugging optimization skills, and comprehensive problem-solving abilities, making the application of computational thinking more sufficient and robust.
- Innovation Layer: Connect with Cutting-edge Needs, Stimulate Innovative Thinking: Students, based on their interests and the latest technological trends, choose their own topics for innovative practice expansion in their spare time, with teachers providing appropriate guidance and necessary resources. Content for innovative practice is specific and clear: in-depth application and personalized configuration of domestic operating systems (openEuler, UnionTech UOS), development of simple system tools based on Python, simple application deployment using container technology (Docker), etc. More importantly, students are encouraged to combine innovative practice with academic competitions and research projects, thereby naturally cultivating innovative design capabilities and cross-scenario application thinking, effectively expanding the boundaries of computational thinking and correspondingly enhancing their core competitiveness.

(4) Reform the Assessment System to Comprehensively Measure Computational Thinking Ability

Breaking away from the single traditional assessment model, a binary assessment system of "process-oriented evaluation (30%) + summative evaluation (70%)" is systematically established. This naturally and appropriately examines students' knowledge system and thinking application abilities, truly achieving "promoting learning and thinking through assessment," and effectively evaluating students' computational thinking, practical abilities, and comprehensive literacy. Crucially, the assessment process always focuses on students' thinking processes. Therefore, in

experimental reports, assignments, and written exam answers, students are required to provide clear and rigorous explanations of problem analysis, solution design, and problem-solving approaches, not just the final answer. Specifically, experimental reports must clearly state objectives, principles, and steps, focusing on problems encountered, analysis processes, and solutions. Written exams directly assess students' ability to solve complex problems using computational thinking, truly realizing "promoting thinking and ability through assessment."

(5) Strengthen Course Resource Construction to Support Computational Thinking Cultivation

Course resources are a vital support for cultivating computational thinking. Based on the course construction objectives, resources are strengthened from aspects of textbooks, digital resources, and teaching staff to provide a solid guarantee for computational thinking cultivation.

- Textbook and Teaching Material Construction: Select the core textbook Computer Operating Systems (4th Edition) by Tang Xiaodan, published by People's Posts and Telecommunications Press, which is comprehensive, logically rigorous, and suitable for computer major teaching and self-study. Aligned with the course reform direction and computational thinking cultivation requirements, supplementary teaching materials are developed, incorporating computational thinking guidance, cutting-edge technology cases, practical guidance, and thinking expansion questions, making up for the textbook's shortcomings regarding insufficient frontier content and lack of thinking guidance, achieving deep adaptation between the textbook and course reform.
- Digital Resource Construction: Build an online teaching platform to systematically upload various digital resources such as teaching courseware, lab manuals, preview materials, and thinking expansion questions. Naturally and appropriately introduce national-level quality MOOC resources to supplement teaching content. Simultaneously, create an online Q&A community to promptly solve problems encountered by students during learning, thereby naturally creating a favorable atmosphere for autonomous learning and mutual assistance.
- Faculty Development: Through enhanced teacher training, systematically organize teachers to participate in various training sessions and seminars on Emerging Engineering teaching reform, computational thinking cultivation, and mainstream operating system applications. Actively encourage teachers to engage in enterprise internships to understand cutting-edge industry technologies and talent demands, naturally introducing real industry cases into classroom teaching, thereby effectively promoting course teaching reform.

5. Teaching Practice and Effectiveness Analysis

This course reform plan has been effectively implemented among over 370 students across seven classes of the 2022 and 2023 cohorts in Computer Science and Technology at our university. The total course hours are 64, consisting of 48 lecture hours and 16 practical hours. Therefore, the proposed course construction path for advancing teaching reform could be naturally and appropriately followed. The selected core textbook was used alongside supporting materials. Multiple teaching methods were adopted, a layered practice system established, and assessment methods reformed, truly putting the reform measures into practice.

Due to the robust implementation of teaching practice, the course reform has been highly successful. Students' learning initiative, computational thinking ability, practical skills, and various comprehensive competencies have been effectively enhanced. Teaching quality and educational outcomes have significantly improved.

From the current description, the effectiveness is evident in four main aspects: First, students' learning initiative has greatly increased. Due to the use of diverse teaching methods and the design of layered practical projects, 92% of students found the course excellently combined theory with practice and very practical, and 88% actively participated in class discussions and practical projects, resulting in high learning enthusiasm and effectively improved autonomous learning ability. Second, students' computational thinking and practical abilities have made significant progress. Students can independently and methodically complete comprehensive practical tasks and naturally apply computational thinking to solve complex problems. Consequently, the number of awards in subject competitions has substantially increased compared to before the reform. Third, course teaching quality has significantly improved. The excellence and pass rates in student assessments have greatly increased, and course satisfaction has reached over 90%. This naturally and convincingly lays a solid foundation for students' subsequent learning of professional courses such as Compiler Principles and Embedded Systems. Fourth, the effectiveness of industry-education integration is highly prominent. The employment rate of graduates in positions such as system development and operations increased by 20% compared to pre-reform levels, and their practical abilities and comprehensive competencies have received unanimous praise from employers.

6. Conclusion

Although the curriculum reform has achieved remarkable results, it is undeniable that there are several issues in its implementation that deserve attention and optimization, and targeted improvements will be made in the follow-up. For the problem that students with weak foundations have great difficulties in designing practical tasks and cannot smoothly adapt to the requirements of the hierarchical practical system, we will strengthen the counseling for this group, design targeted practical tasks and counseling plans, thereby naturally and solidly improving their practical operation ability and thinking level; regarding the problem that the depth and breadth of cutting-edge technology content need to be reasonably balanced, we will optimize the curriculum knowledge system, reasonably control the depth and breadth of cutting-edge technology content, make basic theories and cutting-edge technologies blend naturally and harmoniously, and realize the proper integration of cutting-edge knowledge and basic content; for the problem that the evaluation indicators of computational thinking are not specific enough, making it difficult to objectively and reliably measure the extent of students' thinking improvement, we will clarify the relevant evaluation indicators, establish quantifiable and operable evaluation standards, and objectively and accurately assess the situation of students' thinking improvement; aiming at the problem that some students' practical operation ability of the operating system is insufficient, which directly affects the practical effect and subsequent thinking training, we will

appropriately increase the proportion of class hours for practical operation training, select typical practical operation cases, and effectively improve students' proficiency in operating system practical operations.

Acknowledgments

The authors gratefully acknowledge the financial support from Teaching Demonstration Course Project of Education Department of Anhui Province: Principles of Operating System (2020SJJXSFK1294), Model Course for Ideological and Political Education of Anhui Xinhua University: Principles of Operating System (2023kcszx04), Anhui Provincial Quality Engineering Project (2024cxtd162).

References

[1] Zhao Huiling, Meng Xianying, Mao Yingshuang. "Research and Practice on Teaching Reform of Computer Major under the Background of 'Emerging Engineering Education'." Heilongjiang Education (Theory and Practice), 2021, (09): 72-73.

[2] Yan Qingzhuo, Lin Lina. "Reconstruction of Teaching Content System for Operating System Principles Course from the Perspective of Emerging Engineering Education." Science and Education Guide, 2021(12): 4.

[3] Zhou Junhai. "Research and Practice of Operating System Teaching Reform Based on OBE Education Model under the Background of Emerging Engineering Education." Software Engineering, 2020, 23(04): 51-53.

[4] Ge Wenyi, Gong Jinnan, Pan Guanghui. "Exploration of Teaching Reform in Linux Operating System Programming for Cultivating Xinchuang Talents." Computer Knowledge and Technology, 2024, 20(21): 142-144.