**Scheduling Simulator**

**Question 1:**

З 2 процесами: Система буде чергувати між двома процесами, дозволяючи кожному з них працювати по 2000 мс, а потім блокуватися на 500 мс. Чергування між ними має бути досить рівномірним.

З 5 процесами: Чергування між більшою кількістю процесів може призвести до меншої кількості часу, виділеного кожному процесу в одному циклі, збільшуючи частоту переключень.

З 10 процесами: Це буде ще більш виразно. Збільшення кількості процесів може вплинути на ефективність і загальний прогрес кожного процесу, оскільки вони будуть конкурувати за процесорний час.  
  
**Deadlock Simulator**

**Question 1:**

Команда java deadlock a 2 2 не призводить до взаємного блокування (deadlock), оскільки є достатня кількість ресурсів (2 інстанції) для обох процесів (2 процеси), що запобігає конфлікту за ресурси. Кожен процес має доступ до двох ресурсів, тому вони можуть виконуватися паралельно без взаємного блокування.

**Question 2:**

При виконанні команди java deadlock b 2 1 1, відбувається взаємне блокування, оскільки обидва процеси виконують запити на ресурси, які вже зайняті іншим процесом, що призводить до ситуації, де кожен процес чекає на ресурс, зайнятий іншим. Через обмежену кількість ресурсів (лише один екземпляр кожного ресурсу) процеси не можуть виконуватися незалежно і в результаті блокують один одного.

**Question 3:**

Команда java deadlock b 2 1 2 не призводить до взаємної блокування, оскільки наявність додаткового ресурсу дозволяє уникнути конфліктів за ресурси. Наявність цього додаткового ресурсу робить можливим для процесів працювати паралельно без взаємного блокування.

**Memory Management Simulator**

**Question 1:**

При аналізі симуляції управління пам'яттю, ми створили командний файл для відображення 8 фізичних сторінок на перші 8 віртуальних сторінок, а потім читали з однієї адреси віртуальної пам'яті на кожній з 64 віртуальних сторінок. Це дозволило нам відстежити, які віртуальні адреси викликали помилки сторінок. Ми встановили, що первинний алгоритм заміни сторінок, який використовувався в симуляторі, може бути алгоритмом FIFO або LRU. Потім, модифікуючи replacePage() у PageFault.java, ми реалізували Round Robin і LRU алгоритми. За Round Robin, сторінки замінюються циклічно, тоді як LRU вибирає для заміни сторінку, що використовувалась найдавніше, покращуючи ефективність використання пам'яті.

**File System Simulator**

**Question 1:**

Обмеження каталогів, які ми створимо перед заповненням файлової системи, визначається розміром записів каталогу, кількістю i-вузлів і загальною ємністю сховища.

**Question 2:**

Програма find.java, буде перевіряти існування заданого шляху і, у разі його наявності, виводити імена цього шляху та всіх файлів у всіх директоріях і піддиректоріях під ним. Використання рекурсивного методу дозволить програмі ефективно пройти кожну директорію та її піддиректорії. Максимальна глибина дерева директорій, на яку програма буде працювати, залежить від обмежень стека викликів та системних ресурсів. При перевищенні цих обмежень, програма може зіткнутися з помилкою переповнення стека.

**Question 3:**

**План розробки:**

1. Kernel.chown(): Змінює UID і GID файлу. Тільки власник файлу або суперкористувач може змінювати GID, а тільки суперкористувач може змінювати UID. Протестовано за допомогою програм chown.java і chgrp.java.
2. Kernel.chmod(): Змінює режим доступу до файлу. Тільки власник або суперкористувач може змінити режим, при цьому змінюються лише 9 нижніх бітів режиму. Протестовано з chmod.java.
3. Kernel.umask(): Зберігає umask у контексті процесу. Kernel.creat() тепер використовує umask для визначення режиму нових файлів.
4. Kernel.link(): Створює жорстке посилання на файл. Протестовано за допомогою ln.java.
5. Kernel.unlink(): Видаляє файл або зменшує кількість посилань на файл. Протестовано з rm.java.
6. Kernel.access(): Перевіряє, чи може поточний процес доступитися до файлу у заданому режимі.
7. Система тепер відстежує час останнього доступу, модифікації та зміни статусу (atime, mtime, ctime) для файлів та директорій.
8. Додано підтримку непрямих, подвійних та потрійних непрямих блоків.
9. Підтримка спрощеної системи спільного використання файлів: файл не можна відкрити для запису, якщо він вже відкритий іншим процесом.
10. Можливість використання списку замість бітової карти для відстеження вільних даних блоків.
11. fsck.java: Перевіряє внутрішню консистентність файлової системи.