Міністерство освіти та науки України
Національний університет “Одеська політехніка”
Інститут комп'ютерних систем
Кафедра "Комп'ютеризовані системи управління"

# Пояснювальна записка до курсової роботи

## з курсу «Сучасні технології програмування»

Виконав:
Ст. гр. АТ-191
Греков Д.Е.
Перевірив:
Сперанский В.А.

Одеса 2021

# Оглавление

# Введение

Данная курсовая работа на тему: «Обслуживание процессором ЭВМ очереди готовых заданий» является теоретической, ориентированной на изучение концепций построения операционных систем и методов управления ресурсами процессора.

Целью курсовой работы является изучение основных методов, используемых при управлении ресурсами в различных операционных системах.

Задачей курсовой работы является получение, как теоретических знаний, так и практических навыков, достаточных для проектирования и программирования системного программного обеспечения современных компьютеров, ознакомление с проблемами моделирования и анализа эффективности функционирования реальных вычислительных систем.

# Постановка задачи

Для модели вычислительной системы (ВС) с N-ядерным процессором и мультипрограммным режимом выполнения поступающих заданий требуется разработать программную систему для имитации процесса обслуживания заданий в вычислительных системах.

При построении модели функционирования вычислительной системы должны учитываться следующие основные моменты обслуживания заданий:

- генерация нового задания;

- постановка задания в очередь для ожидания момента освобождения процессора;

- выборка задания из очереди при освобождении процессора после обслуживания очередного задания.

**Генерация задания**:

Считается, что в распоряжении вычислительной системы имеется N ГБ оперативной памяти для размещения рабочей области процесса и M (3<=m<=5) ресурсов R1, R2,…, Rm, обращение к которым переводит процесс в состояние ожидания.

Генерация нового задания (процесса) может происходит автоматически системой как случайное событие.

Каждый процесс характеризуется:

- именем;
- длиной рабочей области;
- интервалом непрерывного выполнения;
- причиной прекращения непрерывной работы (обращение к ресурсу или завершение работы);
- приоритетом, если он требуется используемым методом

планирования процессора.

Перед постановкой задания в очередь имитируется размещения рабочей области процесса в оперативной памяти. В случае невозможности размещения процесс отвергается, в противном случае ему выделяется память и процесс помещается в очередь готовых заданий.

Размещение в ОП происходит методом первого подходящего.

Выборка задания из очереди готовых процессов происходит в момент, когда текущий процесс исчерпал интервал непрерывной работы и освободил CPU. В случае обращения к ресурсу процесс помещается в очередь к нему, причем время использования ресурса генерируется случайным образом. В случае завершения процесс удаляется из очереди готовых процессов.

Стратегия планирования процессора согласно варианту — HPF (Highest Priority First) с вытеснением и неупорядоченной очередью. При появлении процесса с более высоким приоритетом текущий процесс прерывается и управление переходит к вновь прибывшему процессу. Вытесненный процесс возвращается в очередь готовых процессов.

Каждый раз, когда процессор освобождается, из очереди готовых процессов должно быть извлечено задание с наибольшим приоритетом. Очередь просматривается каждый раз, когда нужен новый процесс, так как она неупорядочена.

# Разработка приложения

Для построения программной модели имитации процесса обслуживания заданий в вычислительной системе построим объектную модель этой системы.

**Классы:**

1. Процесс (Process)
2. Утилиты (Utils)
3. Общая очередь процессов (Queue)
4. Очередь готовых процессов (ProcessQueue)
5. Очередь отказаных процессов (RejectsQueue)
6. Блоки памяти (MemoryBlock)
7. Планировщик памяти (MemorySheduler)
8. Процессор (CPU)
9. Ядро (Core)
10. Конфигурация (Configuration)
11. Тактовый генератор (ClockGenerator)
12. Планировщик (Scheduler)

**Перечисления:**

1. Состояния (State)

Опишем каждый класс подробнее:

1. Процесс (Process)
   1.1 Поля:
   - идентификационный номер процесса (id);
   - имя (name);
   - интервал работы в тактах (time);
   - время поступления процесса в систему (timeIn);
   - время работы процесса на центральном процессоре (burstTime);
   - состояние процесса (state);
   - приоритет (priority);
   - требуемый объем памяти (memory);

   1.2 Методы:
   - конструктор Process с параметром id (устанавливает начальное значение параметров процесса);
   - визуализация toString (отображает данные о процессе);

2. Утилиты (Utils)
   2.1 Поля:
      • объект класса Random (random);

   2.2 Методы:

      • геттер getRandomInteger с параметром size (формирует случайное число в пределах от 0 до (size+1));
      • геттер getRandomInteger с параметрами left и right (формирует случайное число в пределах от числа left до числа (right - left + 1));
3. Общая очередь процессов (Queue)
   3.1 Поля:
      • очередь процессов (queue);
      • идентификационный номер последнего процесса (lastId);
   3.2 Методы:
      • геттер;
      • конструктор;
      •добавление в очередь add (создает новый процесс с идентификационным номером на один больше предыдущего и добавляет его в очередь);
      • добавление в очередь add с параметром N (добавляет в очередь N новых процессов);
      • визуализация toString (отображает данные об очереди);
4. Очередь готовых процессов (ProcessQueue)
   Наследник класса Queue
   4.1 Поля:
      • очередь отказаных процессов
   4.2 Методы:
      •добавление в очередь add (переопределенный методо класса Queue);
5. Очередь отказаных процессов (RejectsQueue)
   Наследник класса Queue
6. Блоки памяти (MemoryBlock)
   6.1 Поля:
      • начальное значение блока (start);
      • конечное значение блока (end);
   6.2 Методы:

- компаратор byEnd (сравнивает блоки памяти по конечному значению)
- конструктор;
- визуализация toString (отображает данные о блоке памяти);

7. Планировщик памяти (MemorySheduler)
   7.1 Поля:
   - очередь блоков памяти (memoryBlocks);

   7.2 Методы:
   - поиск свободного блока findFreeBlock (поиск по методу первый подходящий);
   - заполнение свободных блоков fillMemoryBlock (вызывает метод findFreeBlock);
   - освобождение блока памяти releaseMemoryBlock с параметром memoryBlock (удаляет указанные блок памяти из очереди);
   - добавление блока памяти add с параметром memoryBlock (добавляет указанный блок памяти в очереди);
   - визуализация toString (отображает данные об очереди блоков памяти);

8. Процессор (CPU)
   8.1 Поля:
   - массив ядер (cores);

   8.2 Методы:
   - геттер;
   - сеттер;
   - конструктор;
   - визуализация toString (отображает данные о состоянии ядра);

9. Ядро (Core)
   9.1 Поля:
   - состояние ядра (isFree);

   9.2 Методы:
   - геттер;

10. Конфигурация (Configuration)
    10.1 Поля:
    - объем памяти (memoryVolume);
    - максимальный приоритет (maxpriority);
    - минимальный объем памяти процесса (leftBorder);
    - минимальный объем памяти процесса (rightBorder);

11. Тактовый генератор (ClockGenerator)

Наследник класса TimerTask.

11.1 Поля:

- время (time);

11.2 Методы:

- геттер;
- увеличение времени incTime (увеличивает время на единицу);
- увеличение времени Time с параметром tact (увеличивает время на заданную величину);
- отсчет времени run (переопределенный метод класса TimerTask);

12. Планировщик (Scheduler)

12.1 Поля:

- очередь готовых процессов (processQueue);
- очередь отказанных процессов (rejectsQueue);
- процессор (cpu);
- планировщик времени (memorySheduler);
- тактовый генератор (clockGenerator);

12.2 Методы:

- конструктор (устанавливает поля класса и вызывает метод init);
- заполнение очередей init (заполняет очередь блоков памяти и очереди процессов);
- визуализация toString (отображает данные о состоянии ядра);

## Практическое применение

Полученное приложение автоматически генерирует заданное число процессов, добавляет их в очередь готовых или отказанных процессов в зависимости от объема памяти процессора и объема памяти, требуемого для данного процесса, а также выводит данные о процессах и состоянии процессора, ядер и блоков памяти.

# Выводы

При разработке данного приложения были изучены основные стратегии планирования процессора, в том числе заданная по варианту задания стратегию HPF с вытеснением из неупорядоченного списка.

Был реализован метод планирования памяти первый подходящий, в соответствии с которым процесс помещается в первый подходящий по размеру блок памяти.

В итоге было получено консольное приложение, которое автоматически генерирует заданное число процессов, добавляет их в очередь готовых или отказанных процессов в зависимости от объема памяти процессора и объема памяти, требуемого для данного процесса, а также выводит данные о процессах и состоянии процессора, ядер и блоков памяти.

# Список литературы

1. Рекомендации к выполнению курсовой работы – Одесса, 2019. – 36 стр.
2. Сетевые операционные системы / В. Г. Олифер, Н. А. Олифер. – СПб.: Питер, 2002. – 544 с.
3. Д. Цикритзис, Ф. Бернстайн. Операционные системы / пер. с англ. – М.: Мир, 1977. –336с.
4. П. Кейлингерт. Элементы операционных систем. Введение для пользователей / пер. с англ. –М.: Мир, 1985. -295с.
5. А. Шоу. Логическое проектирование операционных систем / пер. с англ. –М.: Мир, 1981. –360 с.
6. Таненбаум Э., Вудхалл А. Операционные системы. Разработка и реализация (+CD). Классика CS. 3-е изд. — СПб.: Питер, 2007. — 704 с: ил.
7. Ахо А., Хопкрофт Д., Ульман Д. – Структуры данных и алгоритмы.: Пер. с англ.: Уч. пос.- М., Издательский дом «Вильямс», 2016. – 400 с.