Федеральное государственное автономное образовательное учреждение высшего образования «Московский физико-технический институт (национальный исследовательский университет)»

УТВЕРЖДЕНО
Проректор по учебной работе и довузовской подготовке
А.А. Воронов

Рабочая программа дисциплины (модуля)

по дисциплине: Теория и технология программирования
по направлению: Системный анализ и управление
профиль подготовки: Системный анализ и управление в технических, экономических и социальных системах
Физтех-школа Аэрокосмических Технологий
кафедра информатики и вычислительной математики
курс: 2
квалификация: бакалавр

Семестры, формы промежуточной аттестации:
3 (осенний) - Экзамен
4 (весенний) - Экзамен

Аудиторных часов: 165 всего, в том числе:
лекции: 90 час.
семинары: 0 час.
лабораторные занятия: 75 час.

Самостоятельная работа: 180 час.

Подготовка к экзамену: 60 час.

Всего часов: 405, всего зач. ед.: 9

Количество контрольных работ, заданий: 8

Программу составили:

В.П. Иванников, д-р физ.-мат. наук, профессор, заведующий кафедрой
В.Е. Карпов, канд. физ.-мат. наук, доцент, доцент
К.А. Коньков, канд. физ.-мат. наук, доцент, заместитель заведующего кафедрой
А.Г. Тормасов, д-р физ.-мат. наук, профессор, заведующий кафедрой

Программа обсуждена на заседании кафедры информатики и вычислительной математики 02.06.2020

Аннотация

Обучающиеся по дисциплине "Теория и технология программирования» изучат историю эволюции вычислительных систем, основные функции, выполняемые современными операционными системами, принципы их внутреннего построения. Будут рассмотрены концепции процессов в операционных системах, основные алгоритмы планирования процессов, логические основы взаимодействия процессов. Студенты изучат программные алгоритмы организации взаимодействия процессов и предъявляемые к ним требования и основные механизмы синхронизации в операционных системах.

1. Цели и задачи

Цель дисциплины

Освоение студентами знаний в области построения и функционирования современных операционных систем и в области разработки современных приложений. Осмысленное применение полученных знаний при изучении других дисциплин.

Задачи дисциплины

- формирование понимания процессов, происходящих в вычислительной системе при запуске и работе программ и программных систем, принципов корректной передачи информации между ними и их взаимной синхронизации;
- обучение студентов методам создания корректно работающих и взаимодействующих программ с помощью системных вызовов операционных систем;
- формирование способности производительно использовать современные вычислительные системы при изучении других дисциплин и при выполнении исследований студентами в рамках выпускных работ на степень бакалавра.

2. Перечень формируемых компетенций

Освоение дисциплины направлено на формирование следующих компетенций:

Код и наименование компетенции	Индикаторы достижения компетенции
ОПК-3 Способен применять полученные знания, умения и навыки для решения типовых задач управления в технических системах	ОПК-3.1 Владеет основными понятиями и законами теории управления

3. Перечень планируемых результатов обучения по дисциплине (модулю)

В результате освоения дисциплины обучающиеся должны

знать:

- историю эволюции вычислительных систем, основные функции, выполняемые современными операционными системами, принципы их внутреннего построения;
- концепцию процессов в операционных системах;
- основные алгоритмы планирования процессов;
- логические основы взаимодействия процессов;
- концепцию нитей исполнения и их отличие от обычных процессов;
- программные алгоритмы организации взаимодействия процессов и предъявляемые к ним требования;
- основные механизмы синхронизации в операционных системах;
- организацию управления оперативной памятью использующиеся при этом алгоритмы;
- основные принципы управления файловыми системами;
- организацию управления устройствами ввода-вывода на уровне как технического, так и программного обеспечения, основные функции подсистемы ввода-вывода;
- принципы сетевого взаимодействия вычислительных систем и построения работы сетевых частей операционных систем;
- основные проблемы безопасности операционных систем и подходы к их решению.
- идеологию объектно-ориентированного подхода;
- принципы программирования структур данных для современных программ;
- типовые решения, применяемые для создания программ.

уметь:

- пользоваться командами командного интерпретатора операционной системы Linux;
- порождать новые процессы, запускать новые программы и правильно завершать их функционирование;
- порождать новые нити исполнения и правильно завершать их функционирование;
- организовывать взаимодействие процессов через потоковые средства связи, разделяемую память и очереди сообщений;
- использовать семафоры и сигналы для синхронизации работы процессов и нитей исполнения;
- использовать системные вызовы для работы с файловой системой;
- разрабатывать программы для сетевого взаимодействия.
- применять объектно-ориентированный подход для написания программ;
- создавать безопасные программы;
- использовать современные средства для написания и отладки программ.

владеть:

- навыками использования команд командного интерпретатора в операционной системе Linux;
- навыками написания и отладки программ, порождающих несколько процессов или нитей исполнения;
- навыками написания и отладки программ, использующих системные вызовы для взаимодействия локальных процессов;
- навыками написания и отладки программ, использующих системные вызовы для работы с файловыми системами и устройствами ввода-вывода;
- навыками написания и отладки сетевых приложений.
- объектно-ориентированным языком программирования (C++, Java, C#);
- средствами использования стандартных библиотек.

4. Содержание дисциплины (модуля), структурированное по темам (разделам) с указанием отведенного на них количества академических часов и видов учебных занятий

4.1. Разделы дисциплины (модуля) и трудоемкости по видам учебных занятий

№	Тема (раздел) дисциплины	Трудоемкость по видам учебных занятий, включая самостоятельную работу, час.			
		Лекции	Семинары	Лаборат. работы	Самост. работа
1	Введение	4		4	12
2	Контрольная работа 1			2	4
3	Контрольная работа 2	2			4
4	Кооперация процессов	8		16	16
5	Проблемы безопасности операционных систем	4			7
6	Процессы и их планирование в операционной системе	8		6	16
7	Сети и сетевые операционные системы	6		4	12
8	Система управления вводом выводом	4		6	16
9	Управление памятью	4		4	8
10	Файловые системы	5		3	10
11	Event-driven и message-driven программирование на примере XWindows Widgets, Mac OS X Interface Builder и подсистемы GDI MS Windows.	2		2	6
12	Адресное пространство приложения: куча, стек и статические объекты.	2		2	4
13	Базовые основы элементарной техники программирования.	4		2	6
14	Безопасность ПО.	2		2	4

15	Динамическая идентификация и приведение типов (RTTI).	2		2	4
16	Краткий обзор ООП реализации в языке C++.	4		2	6
17	Краткий сравнительный обзор ООП реализации в языках C++ и ObjectiveC, позднее и раннее связывание.	4		2	5
18	Параллельное программирование.	4		2	6
19	Принципы и философия ООП в языках, программных системах и операционных системах.	4		4	6
20	Проблемы, специфические для параллельного исполнения многонитевых программ.	4		2	6
21	Процесс написания программ.	4		2	6
22	Работа с разделяемой памятью.	4		2	6
23	Техническая специфика параллельных программ.	4		2	6
24	Эволюция современного аппаратного обеспечения и ее влияние на программное обеспечение.	1		2	4
Итого часов		90		75	180
Подготовка к экзамену		60 час.			
Общая трудоёмкость		405 час., 9 зач.ед.			

4.2. Содержание дисциплины (модуля), структурированное по темам (разделам)

Семестр: 3 (Осенний)

1. Введение

Цели и задачи курса. Понятие о вычислительном комплексе. Системное программное обеспечение и операционные системы. Краткая история эволюции вычислительных систем. Взаимное влияние software и hardware. Автономные, сетевые и распределенные операционные системы. Классификация автономных операционных систем по их назначению и структуре.

Знакомство с операционной системой UNIX. Системные вызовы и библиотека libc. Понятия login и password. Упрощенное устройство файловой системы в UNIX. Полные имена файлов. Текущая директория. Относительные имена файлов. Домашняя директория пользователя. Команда man – универсальный справочник. Команды cd и ls. Перенаправление стандартного ввода и стандартного вывода. Простейшие команды работы с файлами – cat, cp, mkdir, mv, rm. Шаблоны имен файлов. Пользователь и группа. Системные вызовы getuid() и getgid(). Команды chown и chgrp. Права доступа к регулярному файлу и к директории. Команда chmod. Маска создания файлов. Команда umask. Редактирование файлов, компиляция и запуск программ.

2. Контрольная работа 1

Проведение контрольной работы 1

3. Контрольная работа 2

Проведение контрольной работы 2

4. Кооперация процессов

Взаимодействующие и независимые процессы. Категории средств связи. Установление и завершение связи. Прямая и косвенная адресация. Информационная валентность процессов и средств коммуникации. Симплексная, дуплексная и полудуплексная связь. Потоки ввода вывода и сообщения. Буферизация данных. Надежность обмена информацией. Нити исполнения и их отличие от процессов. Interleaving, race condition и взаимоисключения. Условия Бернстайна. Понятие критической секции процесса. Программные алгоритмы организации взаимодействия процессов и предъявляемые к ним требования. Семафоры, мониторы Хора и сообщения.

Понятие потока ввода-вывода в операционной системе UNIX. Работа с файлами через системные вызовы и через функции стандартной библиотеки. Файловый дескриптор. Наследование файловых дескрипторов при системных вызовах fork() и exec(). Системные вызовы open(), read(), write(), close(). FIFO и pipe. Системные вызовы pipe(), mknod(), функция mkfifo(). Особенности системных потоковых вызовов при работе с FIFO и pipe. Преимущества и недостатки потокового обмена данными. IPC в UNIX. Пространство имен. Адресация в System V IPC. Функция ftok(). Дескрипторы System V IPC. Разделяемая память. Системные вызовы shmget(), shmat(), shmdt(), shmctl(). Команды ipcs и ipcrm. Нить исполнения (thread) в UNIX, ее идентификатор. Функция pthread_self(). Создание и завершение нити исполнения. Функции pthread_create(), pthread_exit(), pthread_join(). Семафоры в UNIX. Отличие операций над UNIX семафорами от классических операций. Системные вызовы semget(), semop(), semctl(). Понятие о POSIX семафорах. Очереди сообщений в UNIX. Системные вызовы msgget(), msgsnd(), msgrcv(), msgctl(). Понятие мультиплексирования. Мультиплексирование сообщений. Модель взаимодействия процессов клиент–сервер. Неравноправность клиента и сервера.

5. Проблемы безопасности операционных систем

Классификация угроз. Формализация подхода к обеспечению информационной безопасности. Классы безопасности. Политика безопасности. Криптография как одна из базовых технологий безопасности ОС. Шифрование с симметричными и ассиметричными ключами. Правило Кирхгофа. Алгоритм RSA. Идентификация и аутентификация. Пароли, уязвимость паролей. Авторизация. Разграничение доступа к объектам ОС. Домены безопасности. Матрица доступа. Недопустимость повторного использования объектов. Аудит, учет использования системы защиты.

6. Процессы и их планирование в операционной системе

Понятие процесса. Процесс и программа. Состояния процесса. Управляющий блок процесса и его контекст. Операции над процессами. Переключение контекста. Уровни планирования процессов. Критерии планирования и требования к алгоритмам планирования. Параметры планирования. Вытесняющее и невытесняющее планирование. Алгоритмы планирования: FCFS, RR, SJF, гарантированное планирование, приоритетное планирование, многоуровневые очереди, многоуровневые очереди с обратной связью.

Понятие процесса в UNIX, его контекст. Идентификация процесса. Краткая диаграмма состояний процессов в UNIX. Иерархия процессов. Системные вызовы getpid() и getppid(). Создание процесса в UNIX. Системный вызов fork(). Завершение процесса. Функция exit(). Параметры функции main() в языке C. Переменные среды и аргументы командной строки. Изменение пользовательского контекста процесса. Семейство функций для системного вызова exec().

7. Сети и сетевые операционные системы

Причины объединения компьютеров в сети. Сетевые и распределенные операционные системы. Взаимодействие удаленных процессов как основа работы вычислительных сетей. Многоуровневая модель построения сетевых вычислительных систем. Семейства и стеки протоколов. Эталонная модель OSI/ISO. Удаленная адресация и разрешение адресов. Понятие о DNS. Локальная адресация. Понятие порта. Полные адреса. Понятие сокета (socket). Фиксированная, виртуальная и динамическая маршрутизация. Связь с установлением логического соединения и передача данных с помощью сообщений.

Краткая история семейства протоколов TCP/IP. Общие сведения об архитектуре семейства протоколов TCP/IP. Уровень сетевого интерфейса. Уровень Internet. Протоколы IP, ICMP, ARP, RARP. Internet-адреса. Транспортный уровень. Протоколы TCP и UDP. Понятие порта. Понятие encapsulation. Уровень приложений/процессов. Использование модели клиент–сервер для взаимодействия удаленных процессов. Понятие socket в UNIX. Организация связи между удаленными процессами с помощью датаграмм. Организация связи между процессами с помощью установки логического соединения. Сетевой порядок байт. Функции htons(), htonl(), ntohs(), ntohl(). Функции преобразования IP-адресов inet_ntoa(), inet_aton(). Функция bzero(). Системные вызовы socket(), bind(), sendto(), recvfrom(), accept(), listen(), connect().

8. Система управления вводом выводом

Общие сведения об архитектуре компьютера. Структура контроллера устройства. Опрос устройств и прерывания. Исключительные ситуации и системные вызовы. Прямой доступ к памяти (Direct Memory Access – DMA). Структура системы ввода-вывода. Систематизация внешних устройств и интерфейс между базовой подсистемой ввода-вывода и драйверами. Функции базовой подсистемы ввода-вывода. Блокирующиеся, не блокирующиеся и асинхронные системные вызовы. Буферизация и кэширование. Spooling и захват устройств. Обработка прерываний и ошибок. Планирование запросов. Алгоритмы планирования запросов к жесткому диску: FCFS, SSTF, SCAN, C-SCAN, LOOK, C-LOOK.

Блочные и символьные устройства в UNIX. Понятие драйвера. Блочные, символьные драйверы, драйверы низкого уровня. Файловый интерфейс к драйверам. Коммутатор устройств. Старший и младший номер устройства. Понятие сигнала в UNIX. Способы возникновения сигналов и виды их обработки. Понятия группы процессов, сеанса, лидера группы, лидера сеанса, управляющего терминала сеанса, текущей и фоновой групп процессов. Системные вызовы getpgrp(), setpgrp(), getpgid(), setpgid(), getsid(), setsid(). Системный вызов kill() и команда kill(). Особенности получения терминальных сигналов текущей и фоновой группой процессов. Получение сигнала SIGHUP процессами при завершении лидера сеанса. Системный вызов signal(). Установка собственного обработчика сигнала. Сигналы SIGUSR1 и SIGUSR2. Использование сигналов для синхронизации процессов. Завершение порожденного процесса. Системный вызов waitpid(). Сигнал SIGCHLD и его игнорирование. Возникновение сигнала SIGPIPE при попытке записи в pipe или FIFO, который никто не собирается читать. Понятие о надежности сигналов. POSIX функции для работы с сигналами.

9. Управление памятью

Связывание адресов. Простейшие схемы управления памятью: схема с фиксированными разделами, своппинг, схема с переменными разделами. Проблема размещения больших программ. Понятие виртуальной памяти. Страничная память. Сегментная и сегментно-страничная организации памяти. Таблица страниц. Ассоциативная память. Иерархия памяти. Размер страницы. Исключительные ситуации при работе с памятью. Стратегии управления страничной памятью: выборки, размещения и замещения страниц. Алгоритмы замещения страниц: FIFO, OPT, LRU и другие. Трэшинг (thrashing). Свойство локальности. Модель рабочего множества.

10. Файловые системы

Имена, структура, типы и атрибуты файлов. Операции над файлами. Директории. Операции над директориями. Защита файлов. Методы выделения дискового пространства: непрерывная последовательность блоков, связный список, связный список с индексацией, индексные узлы. Управление свободным и занятым дисковым пространством: битовый вектор, связный список.

Разделы носителя информации (partitions) в UNIX. Логическая структура файловой системы и типы файлов в UNIX. Организация файла на диске в UNIX на примере файловой системы s5fs. Понятие индексного узла (inode). Организация директорий (каталогов) в UNIX. Понятие суперблока. Указатель текущей позиции в файле. Системная таблица файлов и таблица индексных узлов открытых файлов. Операции над файлами и директориями. Понятие жестких и мягких связей. Системные вызовы и команды для выполнения операций над файлами и директориями: chmod, chown, chgrp, open(), creat(), read(), write(), close(), stat(), fstat(), lstat(), ftruncate(), lseek(), link(), symlink(), unlink(). Функции для изучения содержимого директорий opendir(), readdir(), rewinddir(), closedir(). Понятие о файлах, отображаемых в память (memory mapped файлах). Системные вызовы mmap(), munmap(). Понятие виртуальной файловой системы. Монтирование файловых систем в UNIX.

Семестр: 4 (Весенний)

11. Event-driven и message-driven программирование на примере XWindows Widgets, Mac OS X Interface Builder и подсистемы GDI MS Windows.

Event-driven и message-driven программирование на примере XWindows Widgets, Mac OS X Interface Builder и подсистемы GDI MS Windows. Принципы агрегирования COM-объектов. Принципы поддержки ООП в операционных системах и языках, общее и различное. Реализация исключений в языке C++ и в ОС Windows.

12. Адресное пространство приложения: куча, стек и статические объекты.

Адресное пространство приложения: куча, стек и статические объекты. Динамическая инициализация и клонирование объектов. Хранение объектов в адресном пространстве. Виртуальные функции. Наследование абстрактных классов в C++, чисто виртуальные функции.

13. Базовые основы элементарной техники программирования.

Базовые основы элементарной техники программирования. Технические основы программной реализации формальных структур данных, итераторы. Списки, очереди, стеки, наборы, упорядоченные наборы, массивы, деревья, хранение графов, hash-таблицы; примеры использования структур данных, как выбрать структуру, соответствующую задаче. Применение hash-таблиц.

14. Безопасность ПО.

Безопасность ПО. Написание программ без «дырок». Техника кодирования защищенных программ и типичные ошибки. Переполнение буфера, определение уровня доступа, работа с минимально возможными привилегиями, криптография и ее корректное применение, предохранение секретных данных, работа с входными данными, проблемы разных путей доступа к одним и тем же данным, запросов к базам данных и веб-страницам, а также проблемы поддержки интернационального ПО. Моделирование угроз. Классификация опасностей STRIDE – Spoofing (подмена данных), Tampering (подделка и изменение содержания данных), Repudiation (незаконный отказ от проведенной операции), Information disclosure (разглашение информации), Denial of service (отказ в обслуживании), Elevation of privilege (незаконное поднятие привилегий). Методика оценки риска DREAD - Damage potential (что может быть сломано), Reproducibility (повторяемость), Exploitability (пригодность угрозы для использования), Affected users (на каких пользователей повлияет), Discoverability (возможно ли детектировать факт использования).

15. Динамическая идентификация и приведение типов (RTTI).

Динамическая идентификация и приведение типов (RTTI). Обработка исключительных ситуаций. Разные способы «универсализации» алгоритмов – от абстрактных типов данных «ADT» до стандартной библиотеки шаблонов (Standard Template Library) языка программирования C++ (STL).

16. Краткий обзор ООП реализации в языке C++.

Краткий обзор ООП реализации в языках C++. Интерфейсы, полиморфизм и перегрузка операторов в C++. Параметризованные классы. Дружественные функции. Потоки ввода-вывода в C++. Наследование как один из вариантов комбинирования объектов.

17. Краткий сравнительный обзор ООП реализации в языках C++ и ObjectiveC, позднее и раннее связывание.

Краткий сравнительный обзор ООП реализации в языках C++ и ObjectiveC, позднее и раннее связывание. Техническая организация ООП поддержки языков программирования для позднего связывания.

18. Параллельное программирование.

Параллельное программирование. Параллельные программы – от работы с разделяемой памятью, использования массивно-параллельных компьютеров и до распределенных расчетов на многих физических компьютерах. Декомпозиция задач на параллельные куски. Параллелизм данных, параллелизм кода. Паттерны параллельного программирования: параллелизм на уровне задач – декомпозиция задачи, «разделяй и властвуй» – декомпозиция задач и данных, геометрическая декомпозиция – декомпозиция данных, конвейерное исполнение – декомпозиция потока данных, «фронт волны» – декомпозиция данных с «многомерными» зависимостями. Пример типового шаблона программирования – пул нитей.

19. Принципы и философия ООП в языках, программных системах и операционных системах.

Принципы и философия ООП в языках, программных системах и операционных системах. Инкапсуляция, полиморфизм и наследование/агрегирование. Динамический и статический подход в описании классов.

20. Проблемы, специфические для параллельного исполнения многонитевых программ.

Проблемы, специфические для параллельного исполнения многонитевых программ – синхронизация (между несколькими нитями), коммуникация (проблемы полосы пропускания и задержек, связанных с обменом данными между нитями), балансировка нагрузки (между нитями), масштабируемость (эффективность использования многих нитей). Написание высокопроизводительных программ, оценка условий и выбор способов реализации.

21. Процесс написания программ.

Процесс написания программ. Оформление текстов, требования к текстам и комментариям. Сопровождение программ. Документация на ПО, SDP (software development plan – план разработки ПО).

22. Работа с разделяемой памятью.

Работа с разделяемой памятью. Синхронизационные примитивы (низкоуровневые атомарные команды, критические секции, взаимоисключающая блокировка – mutex, рекурсивная блокировка – lock, блокировка чтения-записи – read/write lock, многопроцессорная блокировка – spinlock, семафоры, барьеры). Реализация одних примитивов через другие, относительная «мощность» примитивов. Задача о консенсусе как способ оценки примитивов.

23. Техническая специфика параллельных программ.

Техническая специфика параллельных программ – производительность, условия «гонки» – race condition, взаимная блокировка – deadlock, повторно-входимые программы, потоко-безопасные (thread-safe) библиотеки. Неблокирующие примитивы синхронизации, транзакционная память. Организация высоко-производительных параллельных вычислений.

24. Эволюция современного аппаратного обеспечения и ее влияние на программное обеспечение.

Эволюция современного аппаратного обеспечения и ее влияние на программное обеспечение. Гипертредовые (многопоточные) и многоядерные процессоры, универсальные графические (GPU) процессоры и новые возможности по их использованию.

5. Описание материально-технической базы, необходимой для осуществления образовательного процесса по дисциплине (модулю)

Учебная аудитория, оснащенная мультимедиапроектором и экраном для чтения лекций.
Учебный сетевой компьютерный класс с установленной операционной системой Linux.

6.Перечень рекомендуемой литературы

Основная литература

1. Карпов В. Е., Коньков К. А. Основы операционных систем. – М.: ИНТУИТ.РУ «Интернет-университет информационных технологий», 2011 — 536с.
2. Страуструп Б. Язык программирования C++. – 3-е изд. – М.: БИНОМ, 1999.

Дополнительная литература

1. Олифер В.Г., Олифер Н.А. Сетевые операционные системы. – СПб.: Питер, 2008.
2. Таненбаум Э. Современные операционные системы. – СПб.: Питер, 2017.
3. Руссинович М., Соломон Д. Внутреннее устройство Microsoft Windows: Windows Server 2003, Windows XP, Windows 2000. Мастер-класс. Серия: Мастер-класс. – СПб.: Питер, Русская Редакция, 2008.
4. Лебланк Д., Ховард М. Защищенный код (Writing Secure Code). Второе издание. – Издательство: Русская редакция. Серия: Фундаментальные знания. 2005.
5. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. Приемы объектно-ориентированного проектирования. Паттерны проектирования. – СПб.: ПИТЕР, 2003.

7. Перечень ресурсов информационно-телекоммуникационной сети "Интернет", необходимых для освоения дисциплины (модуля)

1. http://www.intuit.ru,
2. http://cs.mipt.ru

8. Перечень информационных технологий, используемых при осуществлении образовательного процесса по дисциплине (модулю), включая перечень необходимого программного обеспечения и информационных справочных систем (при необходимости)

На лекциях и лабораторных занятиях используются мультимедийные технологии, включая демонстрацию презентаций. На компьютерах в компьютерных классах должна быть установлены операционная система Linux.

9. Методические указания для обучающихся по освоению дисциплины (модуля)

Студент, изучающий курс «Теория и технология программирования» должен, с одной стороны, овладеть общим понятийным аппаратом, а с другой стороны, должен научиться применять теоретические знания на практике.

В результате изучения дисциплины студент должен знать основные определения, понятия, алгоритмы, уметь писать многопроцессные и многонитевые приложения в среде операционной системы Linux, корректно организовывать взаимодействие процессов и нитей, как локальных, так и удаленных, работать с файлами и устройствами ввода-вывода.

Успешное освоение курса требует напряжённой работы студента. В программе курса приведено минимально необходимое время для работы студента над темой. Самостоятельная работа включает в себя:

- чтение и конспектирование рекомендованной литературы;
- проработку учебного материала (по конспектам лекций, учебной и научной литературе), доказательство отдельных утверждений, свойств;
- решение задач, предлагаемых студентам на лекциях и лабораторных работах;
- подготовку к лабораторным работам.
- решение заданий

Руководство и контроль за самостоятельной работой студента осуществляется в форме индивидуальных консультаций.

Показателем владения материалом служит умение решать теоретические и практические задачи. Для формирования умения применять теоретические знания на практике студенту необходимо решать как можно больше практических задач. При решении задач каждое действие необходимо аргументировать, ссылаясь на известные теоретические сведения. Программы должны легко читаться и иметь подробные комментарии.

Важно добиться понимания изучаемого материала, а не механического его запоминания. При затруднении изучения отдельных тем, вопросов, следует обращаться за консультациями к лектору или преподавателю, проводящему лабораторные работы.

ПРИЛОЖЕНИЕ

ОЦЕНОЧНЫЕ МАТЕРИАЛЫ ПО ДИСЦИПЛИНЕ (МОДУЛЮ)

по направлению: Системный анализ и управление
профиль подготовки: Системный анализ и управление в технических, экономических и социальных системах
Физтех-школа Аэрокосмических Технологий
кафедра информатики и вычислительной математики
курс: 2
квалификация: бакалавр

Семестры, формы промежуточной аттестации:
3 (осенний) - Экзамен
4 (весенний) - Экзамен

Разработчики:

В.П. Иванников, д-р физ.-мат. наук, профессор, заведующий кафедрой
В.Е. Карпов, канд. физ.-мат. наук, доцент, доцент
К.А. Коньков, канд. физ.-мат. наук, доцент, заместитель заведующего кафедрой
А.Г. Тормасов, д-р физ.-мат. наук, профессор, заведующий кафедрой

1. Компетенции, формируемые в процессе изучения дисциплины

Код и наименование компетенции	Индикаторы достижения компетенции
ОПК-3 Способен применять полученные знания, умения и навыки для решения типовых задач управления в технических системах	ОПК-3.1 Владеет основными понятиями и законами теории управления

2. Показатели оценивания компетенций

В результате изучения дисциплины «Теория и технология программирования» обучающийся должен:

знать:

- историю эволюции вычислительных систем, основные функции, выполняемые современными операционными системами, принципы их внутреннего построения;
- концепцию процессов в операционных системах;
- основные алгоритмы планирования процессов;
- логические основы взаимодействия процессов;
- концепцию нитей исполнения и их отличие от обычных процессов;
- программные алгоритмы организации взаимодействия процессов и предъявляемые к ним требования;
- основные механизмы синхронизации в операционных системах;
- организацию управления оперативной памятью использующиеся при этом алгоритмы;
- основные принципы управления файловыми системами;
- организацию управления устройствами ввода-вывода на уровне как технического, так и программного обеспечения, основные функции подсистемы ввода-вывода;
- принципы сетевого взаимодействия вычислительных систем и построения работы сетевых частей операционных систем;
- основные проблемы безопасности операционных систем и подходы к их решению.
- идеологию объектно-ориентированного подхода;
- принципы программирования структур данных для современных программ;
- типовые решения, применяемые для создания программ.

уметь:

- пользоваться командами командного интерпретатора операционной системы Linux;
- порождать новые процессы, запускать новые программы и правильно завершать их функционирование;
- порождать новые нити исполнения и правильно завершать их функционирование;
- организовывать взаимодействие процессов через потоковые средства связи, разделяемую память и очереди сообщений;
- использовать семафоры и сигналы для синхронизации работы процессов и нитей исполнения;
- использовать системные вызовы для работы с файловой системой;
- разрабатывать программы для сетевого взаимодействия.
- применять объектно-ориентированный подход для написания программ;
- создавать безопасные программы;
- использовать современные средства для написания и отладки программ.

владеть:

- навыками использования команд командного интерпретатора в операционной системе Linux;
- навыками написания и отладки программ, порождающих несколько процессов или нитей исполнения;
- навыками написания и отладки программ, использующих системные вызовы для взаимодействия локальных процессов;
- навыками написания и отладки программ, использующих системные вызовы для работы с файловыми системами и устройствами ввода-вывода;
- навыками написания и отладки сетевых приложений.
- объектно-ориентированным языком программирования (C++, Java, C#);
- средствами использования стандартных библиотек.

3. Перечень типовых (примерных) вопросов, заданий, тем для подготовки к текущему контролю

3. Перечень типовых контрольных заданий, используемых для проверки знаний, умений, навыков.

1. Примеры задач для лабораторных работ

3 (осенний) семестр

1. Вход в систему Linux. Смена пароля. Использование команд интерпретатора *pwd, ls, cd*. Освоение встроенного справочника *man*. Использование команд *cat, cp, mkdir, mv, rm, chown, chgrp, chmod, umask.* Освоение программы Midnight Commander. Написание, отладка и запуск программы, выводящей идентификатор пользователя и его группы.
2. Прогон и анализ программы, порождающий новый процесс. Написание, отладка и прогон программы с разным поведением ребёнка и родителя. Написание, отладка и прогон программы, использующей аргументы функции *main().* Прогон и анализ программы, изменяющей свой пользовательский контекст. Написание, отладка и прогон программы, порождающей новый процесс, который изменяет свой пользовательский контекст.
3. Прогон и анализ программы, записывающей информацию в файл в потоковом режиме. Написание, отладка и прогон программы, считывающей информацию из файла в потоковом режиме. Прогон и анализ программы, организующей pipe в одном процессе. Прогон и анализ программы, осуществляющей одностороннюю передачу данных через pipe между родителем и ребёнком. Написание, отладка и прогон программы, осуществляющей передачу данных через pip'ы между родителем и ребёнком в двух направлениях. Написание, отладка и прогон программы, определяющей ёмкость буфера pipe. Прогон и анализ программы, осуществляющей одностороннюю передачу данных через FIFO между родителем и ребёнком. Написание, отладка и прогон программы, осуществляющей передачу данных через FIFO между не близкородственными процессами.
4. Прогон и анализ двух программ, взаимодействующих через общую память и печатающих количество собственных запусков. Написание, отладка и прогон двух программ, одна из которых записывает в общую память свой исходный текст, а вторая – распечатывает его из общей памяти. Обнаружение race condition в предыдущих программах. Модернизация программ, считающих количество своих запусков, с помощью алгоритма Петерсона для корректного взаимодействия. Прогон и анализ программы, создающей новую нить исполнения. Написание, отладка и прогон программы, создающей несколько новых нитей исполнения. Написание, отладка и прогон программы, создающей несколько новых нитей исполнения с синхронизацией их работы.
5. Прогон и анализ двух программ, взаимодействующих через семафор (одна программа блокируется, если n раз не была запущена другая программа). Написание, отладка и прогон программы, порождающей новый процесс и осуществляющей полудуплексную связь между родителем и ребёнком через единственный pipe с синхронизацией с помощью семафора(ов). Модернизация программ, считающих количество своих запусков, с помощью семафоров для корректного взаимодействия.
6. Прогон и анализ двух программ, передающих текстовую информацию в одном направлении через очередь сообщений. Написание, отладка и прогон двух программ, передающих текстовую информацию в двух направлениях через единственную очередь сообщений. Написание, отладка и прогон двух программ, передающих смешаную информацию в двух направлениях через единственную очередь сообщений. Написание, отладка и прогон двух программ, осуществляющих взаимодействие клиент-сервер через единственную очередь сообщений.
7. Написание, отладка и прогон программы для определения глубины рекурсии символьной связи. Написание, отладка и прогон программы – аналога команды *ls*. Написание, отладка и прогон двух программ, взаимодействующих через файл, отображаемый в память.

8. Прогон и анализ программы, игнорирующей сигнал SIGINT. Написание, отладка и прогон программы, игнорирующей сигналы SIGINT и SIGQUIT. Прогон и анализ программы, восстанавливающей предыдущую реакцию на сигнал SIGINT. Написание, отладка и прогон программы, восстанавливающей предыдущую реакцию на сигналы SIGINT и SIGQUIT. Написание, отладка и прогон двух программ, обменивающихся числовой информацией через сигналы.
9. Прогон, анализ и модификация сетевых программ клиент-сервер, взаимодействующих через протокол UDP. Прогон, анализ и модификация сетевых программ клиент-сервер, взаимодействующих через протокол TCP.

2. Примеры домашних заданий

3 (осенний) семестр

Задание 1

1. Напишите программу *useless* (*UNIX SYSTEM EXTREMELY LATE EXECUTION SOFTWARE SYSTEM*), которая читает файл и запускает указанные в нем программы с указанной задержкой от времени старта программы *useless*. Формат записи в файле:
<время задержки в секундах> <программа для выполнения>
Файл не является упорядоченным по временам задержки.
2. Для того чтобы не допустить потерю информации при порче диска, обычно используют резервное копирование файлов (*backup*). Простейшей формой *backup*'а является копирование всех файлов из одной директории в другую. Этот способ требует много времени и места на диске. Напишите программу, осуществляющую более интеллектуальный подход.
Программа должна брать из командной строки два параметра: исходную директорию и директорию назначения. Она должна рекурсивно сканировать исходную директорию, делать копии всех файлов, для которых ранее не делались копии или которые были изменены с момента последнего *backup*'а, размещая их в соответствующих местах директории назначения.
После копирования каждого файла должна вызываться команда сжатия *gzip*. Это уменьшит требуемый размер дисковой памяти; а файл будет переименован с добавлением расширения *.gz*. Все возникающие ошибки (нет исходной директории, файл закрыт для чтения и т.д.) должны корректно обрабатываться с выдачей соответствующего сообщения.

Задание 2

1. Напишите программу *runsim*, осуществляющую контроль количества одновременно работающих *UNIX*-приложений. Программа читает *UNIX*-команду со стандартного ввода и запускает ее на выполнение. Количество одновременно работающих команд не должно превышать N, где N – параметр командной строки при запуске *runsim*. При попытке запустить более чем N приложений выдайте сообщение об ошибке. Программа *runsim* должна прекращать свою работу при возникновении признака конца файла на стандартном вводе.
2. На мойке посуды в ресторане работают два человека. Один из них моет посуду, второй вытирает уже вымытую. Времена выполнения операций мытья и вытирания посуды меняются в зависимости от того, что моет. Стол для вымытой, но не вытертой посуды имеет ограниченные размеры.
Смоделируйте процесс работы персонала следующим образом: каждому работнику соответствует свой процесс. Времена выполнения операций содержатся в двух файлах. Каждый файл имеет формат записей:

<тип посуды> : <время операции>

Стол вмещает N предметов независимо от их наименования. Значение N задается как параметр среды *TABLE_LIMIT* перед стартом процессов. Грязная посуда, поступающая на мойку, описывается файлом с форматом записи:

<тип посуды> : <количество предметов>

Записи с одинаковым типом посуды могут встречаться неоднократно.
Организуйте передачу посуды от процесса к процессу:
а) через файл, используя семафоры для синхронизации;
б) через *pipe*;
в) через сообщения;
г) через разделяемую память;
д) через *sockets*.

4 (весенний) семестр

Каждый студент должен сделать по одной задаче из каждого задания.

Требования к оформлению

Текст программы должен быть оформлен профессиональным образом. Отдельно должен быть предоставлен файл (файлы) с текстом решения задачи, тестом на нее (если нужен) и вставляемые файлы заголовков. То есть решение задачи состоит из не менее чем двух файлов (*.h, .CPP*), чаще три (включая тест). Запрещается вставка одного *C/CPP* файла в другой, поощряется использование make-файла или проектов (но не обязательно).
Комментарии в тексте программы: обязаны присутствовать в файлах в начале, должны быть отдельно написаны к каждой функции и в коде по тексту (в среднем каждая третья строка должна содержать комментарий); текст должен выглядеть красиво (отступы и т.д. должны быть оформлены нормально, не должно использоваться более 132 символов в строке), программа не должна иметь неиспользуемых или ненужных кусков (закомментированных). Примером оформления являются *samples*, приложенные к соответствующим дистрибутивам компиляторов (примеры *SDK Windows, Linux kernel sources* и т.д.).

Обязательно:
Необходимо проверять все параметры функций, используя *assert()* или аналоги. Необходимо проверять все коды возврата функций (особенно *malloc* и системных вызовов).

Желательно:
составление для курсовой работы *Software Development Plan* (*SDP*) – по усмотрению преподавателя.

Задание 1

Основная цель – правильное оформление текста задачи, проверка всех требуемых параметров и т.д. Реализовать только на языке С (не С++/C#).

Реализовать одну из структур данных, итератор по ней и тест (три файла: алгоритм, файл заголовков, тест). Тест должен демонстрировать работоспособность структуры. Помните, что требуемых структур может быть много (более одного экземпляра!):

1) односвязный список;
2) двусвязный список;
3) очередь;
4) очередь с приоритетами;
5) стек;
6) набор (*set*);
7) упорядоченный набор с введением функции отношения;
8) массив произвольного размера;
9) массив упорядоченный;
10) бинарное дерево;
11) сбалансированное дерево;
12) хэш-таблица с открытой адресацией (хэшировать текстовые строки произвольной длины);
13) хэш-таблица с цепочками (алгоритм из книги Д. Кнута «Искусство программирования. Сортировка и поиск», хэшировать бинарные данные произвольной длины типа *BLOB – Binary Large Object*);

14) битовый массив (то есть значения принимаются битами, а адресация – по номеру бита);

15) направленный граф (в виде квадратной таблицы отношений $M(i,j) = 1$, если есть путь из i в j узел).

Задание 2

Требования к оформлению – такие же. Программа должна быть выполнена на *C++* и в обязательном порядке должна использовать чужие библиотеки абстрактных типов данных (например, *Borland ClassLib, MS MFC/Atl/Stl, GNU libstdc++, STL* и др.) – любую из них.

Задачи должны быть выполнены на C++ или C# (для платформы *Windows*) с использованием более одной нити или процесса по выбору студента, согласованному с преподавателем. Для этого под *Windows* может быть использована библиотека функций *Win32 API* (или более высокоуровневые библиотеки). Под *Unix*: *POSIX Threads*.

По желанию студента задача может быть дополнена графическим интерфейсом.

Единицу исполнения (нить или процесс) ниже будем называть *вычислителем*.

Задачи с использованием деревьев

Задачи этого раздела используют класс *Tree* для создания древесной структуры данных.

1. Поиск ключа узла дерева

Исходные данные: дерево, каждый элемент которого хранит уникальный, случайно сгенерированный ключ; ключ, узел с которым требуется найти, количество используемых вычислителей.

Цель: написать параллельную программу для поиска данного ключа.

Программа должна создать заданное пользователем дерево, считать количество используемых вычислителей и ключ для поиска, найти элемент дерева, содержащий этот ключ, и вывести информацию об этом узле.

2. Поиск максимального элемента дерева

Исходные данные: дерево, каждый элемент которого хранит уникальный, случайно сгенерированный ключ, количество используемых вычислителей.

Цель: написать параллельную программу для поиска максимального ключа.

Программа должна создать заданное пользователем дерево, считать количество используемых вычислителей, найти элемент дерева, содержащий максимальный из ключей и вывести информацию об этом узле.

Переменная, содержащая максимальный элемент, должна быть общей для всех вычислителей. Доступ к ней – синхронизован.

3. Поиск суммы всех элементов дерева

Исходные данные: дерево, каждый элемент которого хранит уникальный, случайно сгенерированный ключ, количество используемых вычислителей.

Цель: написать параллельную программу для поиска суммы всех ключей дерева.

Программа должна создать заданное пользователем дерево, считать количество используемых вычислителей, найти и вывести сумму всех ключей дерева.

Переменная, содержащая сумму ключей, должна быть общей для всех вычислителей. Доступ к ней – синхронизован.

4. Поиск количества узлов дерева

Исходные данные: дерево, количество используемых вычислителей.

Цель: написать параллельную программу для поиска количества узлов в дереве.

Программа должна создать заданное пользователем дерево, считать количество используемых вычислителей, найти и вывести количество всех узлов в дереве.

Переменная, содержащая максимальный элемент, должна быть общей для всех вычислителей. Доступ к ней – синхронизован.

Возможные алгоритмы распараллеливания работы для задач 1–4:

a. Пусть есть N вычислителей. У первого – ссылка на корень дерева. Он берет одно из поддеревьев для поиска себе, остальные раздает свободным вычислителям. Далее вычислители поступают точно так же. Когда свободных вычислителей не остается, каждый выполняет обычный поиск по своему поддереву.

b. Пусть есть N вычислителей, и выбран вычислитель-мастер. Он сам проводит поиск ключа по дереву в ширину до тех пор, пока не найдет N независимых поддеревьев, после чего раздает их для поиска остальным вычислителям.

Усложнение для задач 1–3: пусть каждый узел дерева содержит дополнительную информацию – количество узлов под ним. К основной цели задачи добавляется равномерное распределение загрузки вычислителей: каждый должен получить поддерево (поддеревья) с примерно одинаковым количеством узлов (отличающимся не более чем на единицу).

Вычислительные задачи

5. Вычисление интеграла численными методами

Исходные данные: функция одной переменной, отрезок интегрирования, количество вычислителей.

Цель: написать параллельную программу для вычисления интеграла от данной функции по данному отрезку.

Программа должна считать функцию, отрезок интегрирования, количество вычислителей, посчитать интеграл в несколько вычислителей, вывести результат.

Предполагаемый алгоритм распараллеливания: использовать параллелизм по отрезку. Каждый вычислитель считает интеграл по своей части отрезка с помощью одного из стандартных методов (трапеции, Симпсона, Ньютона–Лейбница и т.д.). Результаты отдельных вычислителей суммируются.

Численный метод выбирается преподавателем.

6. Вычисление интеграла методом Монте-Карло

Исходные данные: функция одной переменной, отрезок интегрирования, количество вычислителей.

Цель: написать параллельную программу для вычисления интеграла от данной функции по данному отрезку методом Монте-Карло.

Программа должна считать функцию, отрезок интегрирования, количество вычислителей, посчитать интеграл в несколько вычислителей, вывести результат.

Предполагаемый алгоритм распараллеливания:

a) Использовать параллелизм по отрезку. Каждый вычислитель считает интеграл по своей части отрезка. Результаты отдельных вычислителей суммируются.

b) Использовать параллелизм по количеству испытаний в методе Монте-Карло. Необходимое количество испытаний делится между N вычислителями. Результаты испытаний обобщаются.

7. Вычисление n-мерного интеграла от функции $f(x_1,x_2,\ldots,x_n) = f_1(x_1)\ldots f_n(x_n)$ методом Монте-Карло

Исходные данные: функция одной переменной, отрезок интегрирования, количество вычислителей.

Цель: написать параллельную программу для вычисления интеграла от данной функции по данному отрезку методом Монте-Карло.

Программа должна считать функцию, отрезок интегрирования, количество вычислителей, посчитать интеграл в несколько вычислителей, вывести результат.

Вариации:

a) Область имеет вид $[a_1, b_1]\times\ldots\times[a_n, b_n]$.

Предполагаемый алгоритм распараллеливания: использовать параллелизм количеству измерений. Каждый вычислитель просто вычисляет свой интеграл.

b) Область имеет произвольный вид.

Предполагаемый алгоритм распараллеливания: использовать параллелизм количеству измерений. В формуле метода Монте-Карло вычислители-рабочие считают значения функций (расчет может быть громоздким) и отсылают их вычислителю-мастеру, который производит суммирование и вычисление окончательного результата.

8. Параллельное перемножение матриц

Исходные данные: две матрицы $n \times m$ и $m \times n$, количество вычислителей.

Цель: написать программу для перемножения матриц параллельно несколькими вычислителями.

Программа должна считать m, n, две матрицы, количество вычислителей; распараллелить процесс их перемножения.

Предполагаемый алгоритм распараллеливания: поскольку каждый элемент матрицы-результата рассчитывается независимо от других, он может быть определен отдельным вычислителем. Вычислитель-мастер занимается распределением работы между остальными. Он хранит базу свободных вычислителей, куда синхронизовано заносятся не имеющие работы и откуда берутся рабочие для вычисления следующего элемента матрицы.

9. Параллельное вычисление детерминанта матрицы заданного размера (параллелизм по дереву рекурсии)

Исходные данные: матрица $n \times n$, количество вычислителей.

Цель: написать программу для расчета детерминанта матрицы параллельно несколькими вычислителями.

Программа должна считать n, матрицу, количество вычислителей; распараллелить процесс вычисления детерминанта матрицы.

Предполагаемый алгоритм распараллеливания: предполагается вычислять определитель методом Гаусса – приведением к верхнетреугольной матрице и далее перемножением элементов диагонали. Надо придумать способ распределения подзадач по вычислителям, например, получив дерево вычислений, оно же дерево рекурсии. А примеры распараллеливания (раздачи элементов дерева вычислителям) описаны в задаче 1 раздела Задачи с использованием деревьев.

10. Решение уравнения $Ax = b$ (A – матрица) методом Крамера: параллелизм по элементам столбца решения

Исходные данные: невырожденная матрица $n \times m$, столбец $n \times 1$, количество вычислителей.

Цель: написать программу для расчета решения уравнения $Ax = b$ параллельно несколькими вычислителями.

Программа должна считать m, n, матрицу A, столбец b, количество вычислителей; распараллелить решение уравнения $Ax = b$ методом Крамера.

Предполагаемый алгоритм решения. Методом Крамера каждый элемент столбца-результата вычисляется независимо от остальных. Поэтому его можно передать для расчета отдельному вычислителю.

11. Решение дифференциального уравнения $y' + y = f(x)$ методами вычислительной математики

Исходные данные: функция $f(x)$ (правая часть), отрезок, на котором ищется решение, количество вычислителей.

Цель: написать программу для расчета y решения уравнения $y' + y = f(x)$ параллельно несколькими вычислителями.

Программа должна считать правую часть, отрезок интегрирования; распараллелить процесс решения дифференциального уравнения $y' + y = f(x)$ заданным методом вычислительной математики.

Способ распараллеливания: стандартный, по отрезку. Отрезок делится на количество вычислителей, и каждому отдается своя часть.

Используемый метод вычислительной математики выбирается преподавателем.

Сортировка

12. Сортировка слиянием

Исходные данные: массив объектов, функция для их сравнения, количество вычислителей.

Цель: написать программу для упорядочивания массива параллельно несколькими вычислителями.

Предполагаемый алгоритм распараллеливания: при сортировке слиянием массив делится на две равных по размеру части, которые сортируются отдельно и сливаются в один. Таким образом, снова получаем дерево рекурсии. Примеры распараллеливания по дереву см. в задаче 1 раздела *Задачи с использованием деревьев*.

Вывод на экран

13. Контроль за выводом на экран нескольких потоков

Исходные данные: нет.

Цель: добиться с помощью средств синхронизации, чтобы несколько вычислителей выводили информацию на экран в строго определенном порядке.

Программа должна создать 26 потоков, каждый из которых выводит свою букву алфавита в бесконечном цикле. Добиться, чтобы все буквы выводились в алфавитном порядке.

Работа с файлами

14. Копирование файла в несколько потоков

Исходные данные: файл, директория его будущего расположения, количество вычислителей.

Цель: написать программу для копирования файла в другое место параллельно несколькими вычислителями.

Предполагаемый алгоритм распараллеливания:

a) Каждый вычислитель открывает файл отдельно. Они делят файл на равные части, каждый копирует свою часть (читая из копируемого файла и записывая в файл результата блоки данных фиксированного размера).

b) (Для нитей одного процесса.) Файл открывается один раз. Каждый вычислитель также должен скопировать свою часть файла. Но, поскольку файловый указатель общий, доступ к нему должен быть синхронизован. После использования указателя (т.е. копирования очередного блока данных из своей части) вычислитель должен вернуть его в первоначальное положение и снова включиться в борьбу за общий ресурс.

15. Копирование директории вместе со всем ее содержимым (параллелизм по файлам)

Исходные данные: директория для копирования, директория ее будущего расположения.

Цель: написать программу для копирования директории вместе со всем ее содержимым в другое место параллельно несколькими вычислителями.

Предполагаемый алгоритм распараллеливания: снова для операции копирования можно составить дерево рекурсии => распараллеливание по дереву (задача 1 раздела *Задачи с деревьями*).

Моделирование

В данном разделе студентам предлагается смоделировать следующие ситуации:

16. «Тренировка футболистов»

Участвующие вычислители: футболисты.

Общий ресурс: мяч.

Ситуация: команда футболистов отрабатывает индивидуальные действия. Каждый играет за себя. Цель футболиста – завладеть мячом и забить гол. Таким образом, он может выполнять два действия:

a) бороться за мяч.
b) бить по воротам, как только мяч оказался у него.

После удара по воротам вратарь выбивает мяч в поле. Футболист забивает гол с определенной вероятностью. Команда играет определенное время, потом определяется победитель – по забитым мячам.

17. «Семейный автомобиль»

Участвующие вычислители: члены семьи.

Общий ресурс: автомобиль.

Ситуация: семья из 4-х (N) человек: отец, мать, сын и дочь – имеет в распоряжении автомобиль. У каждого из них периодически появляется желание/необходимость использовать его. Пользование длится случайное время. По истечении определенного времени (известного заранее) бензин автомобиля заканчивается, и тот, кто в этот момент пользуется автомобилем, едет его заправлять. Кроме того, сын ответственен за мытье автомобиля. Также по истечению известного заранее времени автомобиль загрязняется. Если сын начинает пользоваться грязным автомобилем, он его моет перед этим.

Итого, все члены семьи «умеют» ездить на автомобиле, заправлять его, ожидать его освобождения. Сын помимо прочего «умеет» мыть автомобиль.

18. «Аэропорт»

Участвующие вычислители: самолет.

Общий ресурс: взлетно-посадочные полосы.

Ситуация: в течение каждого часа в аэропорту появляется 5 самолетов, готовых к вылету, и 5 самолетов, готовых приземлиться (каждый из них в случайное время в пределах этого часа). Чтобы взлететь или приземлиться самолету, требуется 3 минуты. В аэропорте 2 взлетно-посадочные полосы.

19. «Воробьи»

Участвующие вычислители: воробей.

Общий ресурс: хлебные крошки.

Ситуация: бабушка на скамейке периодически (пусть раз в Q минут) бросает воробьям по одной ровно N хлебных крошек. Когда бросают крошку хлеба, первый подлетевший к ней воробей хватает крошку, относит ее в сторону и съедает. За счет этого он пропускает «розыгрыш» следующей крошки. Первоначально у скамейки находится M воробьев. Каждую минуту к стае прилетает еще один воробей. Съев P крошек, воробей наедается и улетает. Чтобы количество воробьев не увеличивалось, должно выполняться соотношение $Q = N/P$.

Итак, воробей может выполнять действия:

- Прилететь (рождение процесса).
- Ждать раздачи крошек.
- Драться за крошку.
- Отлететь с крошкой в сторону и съесть ее.
- Улететь (смерть процесса).

20. «Раздача карт»

Участвующие вычислители: игрок.

Общий ресурс: колода карт.

Ситуация: N игроков тянут карты по одной из лежащей перед ними колоды.

Необходимо обеспечить:

- Очередность доступа к колоде.
- Равномерное распределение карт в колоде.

Можно ради интереса после раздачи посчитать количество очков в вытянутых ими картах и определить победителя.

3. Примеры вопросов и задач для контрольных работ

3 (осенний) семестр

Часть1.

1. Какая техническая база характерна для первого периода вычислительной техники (1945-1955 г.г.)?
    - полупроводники (транзисторы, диоды и т.д.)
    - интегральные микросхемы
    - электронные лампы
2. Что было прообразом современных ОС?
    - компиляторы с символических языков
    - библиотеки математических и служебных программ
    - системы пакетной обработки
3. К чему относится термин спулинг (spooling)?
    - к сбору заданий с одинаковым набором ресурсов в пакеты заданий
    - к организации реального ввода пакета заданий и вывода результатов на отдельных специализированных ЭВМ
    - к организации реального ввода пакета заданий и вывода результатов на том же компьютере, который производит вычисления
4. Планирование заданий стало возможным:
    - с появлением систем пакетной обработки
    - с появлением предварительной записи пакета заданий на магнитную ленту
    - с появлением предварительной записи пакета заданий на магнитный диск
5. Что из ниже перечисленного является мультипрограммными вычислительными системами?
    - система, в которой реализован спулинг (spooling)
    - система, в памяти которой одновременно находится несколько программ. Когда одна из программ ожидает завершения операции ввода-вывода, другая программа может исполняться
    - система, в памяти которой находится несколько программ, чье исполнение чередуется по прошествии определенного промежутка времени
6. Возможность интерактивного взаимодействия пользователя и программы возникает с появлением:
    - систем пакетной обработки
    - мультипрограммных вычислительных систем
    - систем разделения времени
7. Разделение персонала, связанного с разработкой и эксплуатацией ЭВМ, на разработчиков, специалистов по эксплуатации, операторов и программистов произошло:
    - в первый период развития вычислительной техники (1945-55 г.г.)
    - во второй период развития вычислительной техники (1955- начало 60-х г.г.)
    - в третий период развития вычислительной техники (начало 60-х - 1980 г.г.)
8. При доступе к файлу в сетевой ОС пользователь должен знать:
    - только имя файла
    - точное физическое расположение файла на диске
    - имя файла, компьютер, на котором находится файл, и сетевой способ доступа к информации в файле
9. При доступе к файлу в распределенной ОС пользователь должен знать:
    - только имя файла
    - точное физическое расположение файла на диске
    - имя файла, компьютер, на котором находится файл, и сетевой способ доступа к информации в файле
10. Из каких состояний процесс может перейти в состояние *ожидание*?
    - из состояния *рождение*
    - из состояния *готовность*
    - из состояния *исполнение*

11. Из каких состояний процесс может перейти в состояние *исполнение?*
    - из состояния *ожидание*
    - из состояния *готовность*
    - из состояния *рождение*
12. Когда процесс, находящийся в состоянии *закончил исполнение*, может окончательно покинуть систему?
    - по прошествии определенного интервала времени
    - только при перезагрузке операционной системы
    - после завершения процесса-родителя
13. Какие из перечисленных ниже компонентов входят в пользовательский контекст процесса?
    - состояние, в котором находится процесс
    - программный счетчик процесса
    - информация об устройствах ввода-вывода, связанных с процессом
    - содержимое регистров процессора
    - код и данные, находящиеся в адресном пространстве процесса
14. Какие из перечисленных ниже компонентов входят в регистровый контекст процесса?
    - состояние, в котором находится процесс
    - программный счетчик процесса
    - информация об устройствах ввода-вывода, связанных с процессом
    - содержимое регистров процессора
    - код и данные, находящиеся в адресном пространстве процесса
15. Какие из перечисленных ниже компонентов входят в системный контекст процесса?
    - состояние, в котором находится процесс
    - программный счетчик процесса
    - информация об устройствах ввода-вывода, связанных с процессом
    - содержимое регистров процессора
    - код и данные, находящиеся в адресном пространстве процесса
16. При модернизации некоторой операционной системы, поддерживающей только три состояния процессов: *готовность, исполнение, ожидание,* принято решение ввести два новых системных вызова. Один из этих вызовов позволяет любому процессу приостановить жизнедеятельность любого другого процесса (кроме самого себя), до тех пор, пока какой-либо процесс не выполнит второй системный вызов. Сколько новых состояний процессов появится в системе?
    - 1
    - 2
    - 3
17. При модернизации некоторой операционной системы, поддерживающей только три состояния процессов: *готовность, исполнение, ожидание,* решено ввести два новых системных вызова. Один из этих вызовов позволяет любому процессу приостановить жизнедеятельность любого другого процесса (кроме самого себя), до тех пор, пока какой-либо процесс не выполнит второй системный вызов. Сколько новых операций над процессами появится в системе?
    - 2
    - 4
    - 5
18. При модернизации некоторой операционной системы, поддерживающей только три состояния процессов: *готовность, исполнение, ожидание,* решено ввести два новых системных вызова. Один из этих вызовов позволяет любому процессу приостановить жизнедеятельность любого другого процесса (кроме самого себя),

до тех пор, пока какой-либо процесс не выполнит второй системный вызов. Сколько новых переходов из состояния *исполнение* появится в системе?

- 0
- 2
- 4

19. Сколько операций над процессами существует в курсе «Операционные системы семейства UNIX»?
    - 4
    - 6
    - 7
20. Какая операция над процессом в модели, принятой в данном курсе, не имеет парной?
    - создание процесса
    - запуск процесса
    - изменение приоритета процесса
21. В некоторой операционной системе, похожей на UNIX, существует единственный способ порождения нового процесса, который будет являться дубликатом родительского процесса по регистровому и пользовательскому контекстам, с помощью системного вызова fork(). Для замены пользовательского контекста процесса и запуска исполняемого файла с именем a.out применяется системный вызов exec("a.out"). Неопытный программист написал следующую программу:

    void main()
    {int i; for (i = 0; i < n; i++){fork(); fork(); exec("a.out");}while(1);}

    где n - некоторая положительная константа. Сколько процессов будет запущено в операционной системе в результате ее выполнения, если программа a.out не запускает новых процессов?
    - 2
    - 4
    - 2^n
22. В каких случаях производится невытесняющее кратковременное планирование процессов?
    - когда процесс переводится из состояния *исполнение* в состояние *завершил исполнение;*
    - когда процесс переводится из состояния *исполнение* в состояние *ожидание;*
    - когда процесс переводится из состояния *ожидание* в состояние *готовность.*
23. На каких параметрах может основываться долгосрочное планирование процессов?
    - на статических параметрах вычислительной системы;
    - на динамических параметрах вычислительной системы;
    - на статических параметрах процессов;
    - на динамических параметрах процессов.
24. Какие из перечисленных алгоритмов допускают неограниченно долгое откладывание выборки одного из готовых процессов на исполнение?
    - FCFS
    - SJF
    - RR
    - многоуровневые очереди
25. Какие из перечисленных алгоритмов представляют собой частные случаи планирования с использованием приоритетов?
    - FCFS
    - RR
    - SJF
    - гарантированное планирование

26. К какому из перечисленных алгоритмов стремится поведение алгоритма RR по мере увеличения кванта времени?
    - SJF
    - FCFS
    - гарантированное планирование при одном процессе на каждого пользователя.
27. К какому из перечисленных алгоритмов теоретически стремится поведение алгоритма RR по мере уменьшения кванта времени:
    - SJF
    - FCFS
    - гарантированное планирование при одном процессе на каждого пользователя
28. Пусть в вычислительную систему поступают пять процессов различной длительности по следующей схеме:

Номер процесса	Момент поступления в систему	Время исполнения
1	2	4
2	1	3
3	4	5
4	3	2
5	0	9

    Чему равно среднее время ожидания процесса (waiting time) при использовании вытесняющего алгоритма SJF? При вычислениях считать, что процессы не совершают операций ввода-вывода, временем переключения контекста пренебречь.
    - 11.3
    - 5.0
    - 8.4
29. Пусть в вычислительную систему поступают пять процессов различной длительности по следующей схеме:

Номер процесса	Момент поступления в систему	Время исполнения
1	2	4
2	1	3
3	4	5
4	3	2
5	0	9

    Чему равно среднее время ожидания процесса (waiting time) при использовании невытесняющего алгоритма SJF? При вычислениях считать, что процессы не совершают операций ввода-вывода, временем переключения контекста пренебречь.
    - 11.3
    - 5.0
    - 8.4
30. Пусть в вычислительную систему поступают пять процессов различной длительности с разными приоритетами по следующей схеме:

Номер процесса	Момент поступления в систему	Время исполнения	Приоритет
1	3	10	1
2	6	4	0
3	0	4	3
4	2	1	4
5	4	3	2

Чему равно среднее время между стартом процесса и его завершением (turnaroud time) при использовании вытесняющего приоритетного планирования? При вычислениях считать, что процессы не совершают операций ввода-вывода, временем переключения контекста пренебречь. Наивысшим приоритетом является приоритет 0.

- 10.6
- 13.4
- 15.0

31. Какая категория средств связи получила наибольшее распространение в вычислительных системах?
    - сигнальные
    - канальные
    - разделяемая память
32. Какой из вариантов адресации может использоваться для организации передачи информации через pipe?
    - симметричная прямая адресация
    - асимметричная прямая адресация
    - непрямая адресация
33. Какое из перечисленных условий надежности связи не может быть выполнено со стопроцентной гарантией при выполнении остальных условий?
    - не происходит потери информации
    - не происходит повреждения информации
    - не нарушается порядок данных в процессе обмена
34. Сколько процессов могут одновременно использовать одно и то же средство связи, пользуясь симметричной прямой адресацией?
    - 2
    - произвольное количество
    - ответ зависит от того, является ли средство связи дуплексным или симплексным
35. Какие процессы могут обмениваться информацией через FIFO?
    - только процесс, создавший FIFO, и его процесс-ребенок
    - только процессы, имеющие общего родителя, создавшего FIFO
    - произвольные процессы в системе
36. Какие процессы могут обмениваться информацией через pipe?
    - только процесс, создавший pipe, и его непосредственный процесс-ребенок
    - только процессы, имеющие общего родителя, создавшего pipe
    - произвольные процессы в системе
37. В операционных системах, поддерживающих нити исполнения (threads) внутри одного процесса на уровне ядра системы, процесс находится в состоянии *готовность*, если:
    - хотя бы одна нить процесса находится в состоянии *готовность*
    - хотя бы одна нить исполнения находится в состоянии *готовность,* и нет ни одной нити в состоянии *ожидание*
    - хотя бы одна нить процесса находится в состоянии *готовность,* и нет ни одной нити в состоянии *исполнение.*
38. В операционных системах, поддерживающих нити исполнения (threads) внутри одного процесса на уровне ядра системы, наряду с блоками управления процессами (PCB) существуют структуры данных для управления нитями - TCB (Thread Control Block). Укажите, какие данные из перечисленных ниже хранятся, по вашему мнению, в TCB:
    - данные о файлах, используемых процессом
    - указатель стека

- идентификатор пользователя, инициировавшего работу процесса

39. В операционных системах, поддерживающих нити исполнения (threads) внутри одного процесса на уровне ядра системы, наряду с блоками управления процессами (PCB) существуют структуры данных для управления нитями - TCB (Thread Control Block). Укажите, какие данные из перечисленных ниже хранятся, по вашему мнению, в TCB:
    - содержимое регистров процессора
    - данные, описывающие расположение адресного пространства процессора
    - приоритет нити исполнения
40. Рассмотрим две активности, P и Q:

P:	Q:
y=x+2	z=x-3
f=y-4	f=z+1

    Набор из этих двух активностей является:
    - детерминированным
    - недетерминированным
    - детерминированность зависит от значения x
41. Рассмотрим две активности, P и Q:

P:	Q:
y=x+1	z=x-3
f=y-4	f=z+1

    Набор из этих двух активностей является:
    - детерминированным
    - недетерминированным
    - детерминированность зависит от значения x
42. Если для некоторого набора активностей условия Бернстайна не выполняются, то набор активностей является:
    - детерминированным
    - недетерминированным
    - может быть как недетерминированным, так и детерминированным
43. Прием взаимоисключения применяется:
    - для того чтобы у процесса не было критического участка
    - для устранения условия гонки
    - для того чтобы процессы не использовали одни и те же ресурсы.
44. Термин race condition (условие гонки) относится:
    - к набору процессов, совместно использующих какой-либо ресурс
    - к набору процессов, демонстрирующих недетерминированное поведение
    - к набору процессов, для каждого из которых важно завершиться как можно быстрее
45. Термин «критическая секция» относится:
    - к участку процесса с наибольшим объемом вычислительной работы
    - к участку процесса, в котором процесс совместно с другими процессами использует разделяемые переменные
    - к участку процесса, выполнение которого совместно с другими процессами может привести к неоднозначным результатам
46. Какие из условий для организации корректного взаимодействия двух процессов с помощью программного алгоритма выполнены для алгоритма «переменная-замок»:
    - условие взаимоисключения
    - условие прогресса
    - условие ограниченного ожидания

47. Какие из условий для организации корректного взаимодействия двух процессов с помощью программного алгоритма выполнены для алгоритма «строгое чередование»:
    - условие взаимоисключения
    - условие прогресса
    - условие ограниченного ожидания
48. Какие из условий для организации корректного взаимодействия двух процессов с помощью программного алгоритма выполнены для алгоритма «флаги готовности»:
    - условие взаимоисключения
    - условие прогресса
    - условие ограниченного ожидания
49. В функциях-методах мониторов Хора обычно реализовываются:
    - только прологи и эпилоги критических участков
    - критические участки взаимодействующих процессов
    - только различные операции над внутренними переменными монитора (как операции над внутренними переменными класса в ООП)
50. Условные переменные в мониторах Хора обычно используются:
    - для обеспечения взаимоисключения в критических участках кооперативных процессов
    - для обеспечения взаимосинхронизации кооперативных процессов
    - для передачи данных между кооперативными процессами
51. Какие из перечисленных механизмов синхронизации обычно реализуются в вычислительной системе с помощью специальных системных вызовов?
    - семафоры Дейкстры
    - мониторы Хора
    - очереди сообщений
52. Отладка программ, содержащих очень большое количество семафоров, затруднена, так как:
    - требует специального программного обеспечения
    - ошибочные ситуации трудновоспроизводимы
    - для хорошего программиста никаких затруднений не возникает
53. Для чего нужен синхронизирующий процесс при реализации семафоров через очереди сообщений:
    - для удобства реализации
    - для обеспечения взаимной синхронизации кооперативных процессов
    - для обеспечения атомарности операций P и V
54. Рассмотрим механизм синхронизации, называемый бинарными семафорами. Бинарный семафор — это семафор, который может принимать всего два значения: 0 и 1. Операция P для этого семафора выглядит так же, как и для семафора Дейкстры, а операция V заключается в простом присваивании семафору значения 1. Бинарные семафоры:
    - обладают меньшими возможностями, чем семафоры Дейкстры
    - обладают большими возможностями, чем семафоры Дейкстры
    - эквивалентны семафорам Дейкстры
55. На каком уровне иерархии памяти находится программа в процессе выполнения?
    - на магнитном диске
    - в оперативной памяти
    - разные компоненты программы могут находиться на различных уровнях
56. Чем обусловлена эффективность иерархической схемы памяти?
    - скоростью обмена с оперативной памятью
    - принципом локальности
    - количеством уровней в иерархии.

57. На каком этапе осуществляется связывание логического и физического адресных пространств процесса в современных ОС?
    - на этапе выполнения
    - на этапе загрузки
    - на этапе компиляции
58. Внутренняя фрагментация - это:
    - потеря части памяти, выделенной процессу, но не используемой им
    - разбиение адресного пространства процесса на фрагменты
    - потери части памяти в схеме с динамическими (переменными) разделами
59. Возможность организации в больших программах структур с перекрытиями (overlays) обусловлена:
    - наличием в программе большого количества независимых процедур
    - разбиением памяти на несколько фиксированных разделов
    - принципом локальности
60. Что понимается под термином «внешняя фрагментация»?
    - потеря части памяти, не выделенной ни одному процессу
    - потеря части памяти в схеме с фиксированными разделами
    - наличие фрагментов памяти, внешних по отношению к процессу
61. Какая из схем управления памятью подвержена внешней фрагментации?
    - схема с фиксированными разделами
    - схема с динамическими разделами
    - страничная организация
62. Какая из схем управления памятью подвержена внутренней фрагментации?
    - схема с фиксированными разделами
    - сегментная организация
    - страничная организация
63. Таблица страниц процесса - это:
    - структура, используемая для отображения логического адресного пространства в физическое при страничной организации памяти
    - структура, организованная для учета свободных и занятых страничных блоков
    - структура, организованная для контроля доступа к страницам процесса
64. В чем состоит преимущество схемы виртуальной памяти по сравнению с организацией структур с перекрытием?
    - возможность выполнения программ большего размера
    - возможность выполнения программ, размер которых превышает размер оперативной памяти
    - экономия времени программиста при размещении в памяти больших программ
65. Чем запись в таблице страниц в схеме страничной виртуальной памяти отличается от соответствующей записи в случае простой страничной организации?
    - наличием номера страничного кадра
    - наличием бита присутствия
    - наличием атрибутов защиты страницы
66. Какая из рассмотренных ниже схем управления памятью пригодна для организации виртуальной памяти?
    - страничная
    - сегментная
    - схема с фиксированными разделами
    - схема с динамическими разделами
67. Разбиение логического адресного пространства процесса на страницы в страничной схеме управления памятью осуществляется:

- программистом, поскольку он может сделать это самым оптимальным образом
- компилятором, который может выполнить размещение различных компонентов программы на разных страницах
- менеджером памяти ОС и блоком управления памятью незаметно для программиста

68. Вычислите номер страницы памяти и смещение для логического адреса 32768, если размер страницы равен 4Кб. Страницы нумеруются, начиная с 0.
    - 8 и 0
    - 5 и 4096
    - 7 и 0
69. Сколько записей содержится в одноуровневой таблице страниц в системе с 32-разрядной архитектурой и размером страницы 4Кб?
    - 2^{32}
    - 2^{20}
    - 2^{12}
70. Предположим, что для организации двухуровневой таблицы страниц 32-разрядный логический адрес разбивается на три поля: a,b,c. Третье поле — c — это смещение внутри страницы. От чего зависит количество страниц в логическом адресном пространстве процесса?
    - от размера всех трех полей
    - от размера поля c
    - от размера поля a
71. В вычислительной системе со страничной организацией памяти и 32-х битовым адресом размер страницы составляет 8 Мбайт. Для некоторого процесса таблица страниц в этой системе имеет вид

1	0x00000000
2	0x02000000
5	0x06000000
6	0x10000000

    Какому физическому адресу соответствует логический адрес 0x00827432:
    - 0x27432
    - 0x02027432
    - 0x10027432
72. В диком каннибальском племени вокруг котла с пищей спят дикари и повар. Изначально в котле находится N порций мяса. Дикари по очереди просыпаются, берут из котла порцию мяса, съедают его и засыпают снова. Дикарь, не обнаруживший мяса в котле, будит повара. Повар находит добычу и снова готовит N порций, не подпуская никого к котлу во время приготовления, после чего тоже засыпает. Используя семафоры Дейкстры и, при необходимости, разделяемые переменные, постройте корректную модель происходящего, описав поведение каждого из дикарей и повара с помощью отдельных процессов.
73. В диком каннибальском племени вокруг котла с пищей спят дикари и повар. Изначально в котле находится N порций мяса. Дикари по очереди просыпаются, берут из котла порцию мяса, съедают его и засыпают снова. Дикарь, не обнаруживший мяса в котле, будит повара. Повар находит добычу и снова готовит N порций, не подпуская никого к котлу во время приготовления, после чего тоже засыпает. Используя мониторы Хора, постройте корректную модель происходящего, описав поведение каждого из дикарей и повара с помощью отдельных процессов.

74. В диком каннибальском племени вокруг котла с пищей спят дикари и повар. Изначально в котле находится N порций мяса. Дикари по очереди просыпаются, берут из котла порцию мяса, съедают его и засыпают снова. Дикарь, не обнаруживший мяса в котле, будит повара. Повар находит добычу и снова готовит N порций, не подпуская никого к котлу во время приготовления, после чего тоже засыпает. Используя классические очереди сообщений, постройте корректную модель происходящего, описав поведение каждого из дикарей и повара с помощью отдельных процессов.

Часть 2.

75. Известно, что время доступа к оперативной памяти 100 нс, а время доступа к ассоциативной памяти – 10 нс. Частота попаданий в ассоциативную память при обращении к данным (hit ratio) составляет 90%. Чему равно среднее время обращения за данным к памяти для одноуровневой таблицы страниц?
    - 19 нс
    - 120 нс
    - 168 нс
76. Для оповещения вычислительной системы об отсутствии нужной страницы в памяти используется:
    - механизм системных вызовов
    - механизм аппаратных прерываний
    - механизм исключительных ситуаций
77. Какую стратегию управления памятью может реализовать алгоритм выталкивания страниц LRU?
    - стратегию размещения страницы в памяти при наличии списка свободных кадров
    - стратегию упреждающей выборки, когда кроме страницы, вызвавшей исключительную ситуацию, в память также загружается несколько страниц, окружающих ее
    - стратегию замещения
78. Какой результат может иметь анализ бита модификации, входящего в состав атрибутов страницы?
    - уменьшение времени обработки page fault`а ввиду того, что копия страницы уже имеется на диске
    - необходимость коррекции записи о странице в таблице страниц, поскольку содержимое страницы изменено
    - блокировку страницы в памяти для того, чтобы сохранить изменения содержимого страницы в неприкосновенности
79. Какие действия обычно выполняет операционная система в том случае, если процесс обратился к отсутствующей странице, а свободных кадров в наличии нет?
    - приостанавливает этот процесс до тех пор, пока в системе не появятся свободные кадры
    - приостанавливает процесс, находит некоторый занятый кадр основной памяти, перемещает в случае надобности его содержимое во внешнюю память, переписывает в этот страничный кадр содержимое логической страницы из внешней памяти, после чего возобновляет выполнение процесса
    - выгружает процесс на диск
80. Для некоторого процесса, запущенного в вычислительной системе со страничной организацией памяти с использованием LRU алгоритма замещения страниц, выделение процессу 4 кадров памяти приводит к 11 page fault`ам, а выделение 6 кадров памяти – к 9 page fault`ам (вначале все кадры свободны). Какой (какие) вариант(ы) количества page fault`ов для того же процесса и того же количества

кадров может быть получен при использовании OPT алгоритма замещения страниц?

- 12 и 8
- 8 и 7
- 7 и 8
- 9 и 6

81. Для некоторого процесса известна следующая строка запросов страниц памяти 1, 5, 2, 3, 2, 1, 4, 5, 1, 2, 3, 4, 1, 5, 2, 1, 3, 2, 4. Сколько ситуаций отказа страницы (*page fault*) возникнет для данного процесса при использовании алгоритма замещения страниц FIFO (Fist Input Fist Output), если процессу выделено 3 кадра памяти?
    - 15
    - 12
    - 10
82. Для некоторого процесса известна следующая строка запросов страниц памяти 1, 5, 2, 3, 2, 1, 4, 5, 1, 2, 3, 4, 1, 5, 2, 1, 3, 2, 4. Сколько ситуаций отказа страницы (*page fault*) возникнет для данного процесса при использовании алгоритма замещения страниц LRU (the Least Recently Used), если процессу выделено 3 кадра памяти?
    - 15
    - 12
    - 10
83. Для некоторого процесса известна следующая строка запросов страниц памяти 1, 5, 2, 3, 2, 1, 4, 5, 1, 2, 3, 4, 1, 5, 2, 1, 3, 2, 4. Сколько ситуаций отказа страницы (*page fault*) возникнет для данного процесса при использовании алгоритма замещения страниц OPT (optimal), если процессу выделено 3 кадра памяти?
    - 15
    - 12
    - 10
84. Файловая система включается в состав ОС для того, чтобы:
    - эффективно использовать дисковое пространство
    - обеспечить пользователя удобным интерфейсом для работы с внешней памятью
    - повысить производительность системы ввода-вывода
85. Известно, что в большинстве ОС файл представляет собой неструктурированную последовательность байтов и хранится на диске. Какой способ доступа обычно применяется к таким файлам?
    - последовательный
    - прямой
    - индексно-последовательный
86. Для чего по окончании работы с файлом принято выполнять операцию закрытия (close) файла?
    - чтобы освободить место во внутренних таблицах файловой системы
    - чтобы перевести указатель текущей позиции в начало файла
    - чтобы разрешить доступ к файлу другим процессам
87. Входит ли имя каталога, в котором находится файл, в полное имя файла на диске?
    - не входит
    - входит
    - это зависит от того, является данный каталог рабочим
88. Схема выделения дискового пространства связным списком блоков не нашла широкого применения, так как:
    - неэффективно использует дисковое пространство
    - требует большого количества обращений к диску при работе с файлами
    - страдает от внутренней фрагментации

89. Основным преимуществом использования таблицы отображения фалов (FAT) по сравнению с классической схемой выделения связным списком является:
    - сокращение количества обращений к диску
    - повышенная надежность
    - более экономичное использование дискового пространства
90. Большинство файловых систем, поддерживаемых ОС Unix для выделения дискового пространства, использует схему:
    - с индексными узлами
    - связного списка блоков
    - выделения непрерывной последовательности блоков
91. Предположим, что один из файлов в ОС Unix жестко связан с двумя различными каталогами, принадлежащими различным пользователям. Что произойдет, если один из пользователей удалит файл?
    - файл автоматически удалится из каталога второго пользователя
    - содержание каталога второго пользователя не изменится
    - система отменит операцию удаления файла
92. Какие из перечисленных ситуаций возникают синхронно с работой процессора?
    - прерывания
    - исключительные ситуации
    - программные прерывания
93. Какие из перечисленных ситуаций возникают предсказуемо?
    - прерывания
    - исключительные ситуации
    - программные прерывания
94. Какие из перечисленных ситуаций обнаруживаются процессором между выполнением команд?
    - прерывания
    - исключительные ситуации
    - программные прерывания
95. Какие из параметров запроса к жесткому диску обычно учитываются при планировании последовательности запросов?
    - вид операции
    - номер сектора
    - номер цилиндра
    - номер дорожки
96. Какие из вариантов реализации системного вызова read могут прочитать меньше байт, чем запросил процесс?
    - асинхронный
    - блокирующийся
    - неблокирующийся
97. Какие из перечисленных функций базовой подсистемы ввода-вывода могут быть делегированы драйверам?
    - поддержка блокирующихся, неблокирующихся и асинхронных системных вызовов
    - обработка ошибок и прерываний, возникающих при операциях ввода-вывода
    - буферизация и кэширование входных и выходных данных
    - планирование последовательности запросов на выполнение операций ввода-вывода
98. Пусть у нас имеется диск с 80 цилиндрами (от 0 до 79). Время перемещения головки между соседними цилиндрами составляет 1мс. Время же перевода головки с 79-го на 0-й цилиндр составляет всего 10 мс. В текущий момент времени головка

находится на 45-м цилиндре и двигается в сторону увеличения номеров цилиндров. Сколько времени будет обрабатываться следующая последовательность запросов на чтение цилиндров: 10, 6, 15, 71, 1, 62, для алгоритма SSTF (временами чтения цилиндров и смены направления движения пренебречь)?

- 121 мс
- 96 мс
- 59 мс

99. Пусть у нас имеется диск с 80 цилиндрами (от 0 до 79). Время перемещения головки между соседними цилиндрами составляет 1мс. Время же перевода головки с 79-го на 0-й цилиндр составляет всего 10 мс. В текущий момент времени головка находится на 45-ом цилиндре и двигается в сторону увеличения номеров цилиндров. Сколько времени будет обрабатываться следующая последовательность запросов на чтение цилиндров: 10, 6, 15, 71, 1, 62, для алгоритма C-SCAN (временами чтения цилиндров и смены направления движения пренебречь)?
    - 121 мс
    - 96 мс
    - 59 мс
100. Пусть у нас имеется диск с 80 цилиндрами (от 0 до 79). Время перемещения головки между соседними цилиндрами составляет 2 мс. В текущий момент времени головка находится на 23-м цилиндре и двигается в сторону увеличения номеров цилиндров. Сколько времени будет обрабатываться следующая последовательность запросов на чтение цилиндров: 11, 22, 10, 73, 1, 12, алгоритма SCAN (временами чтения цилиндров и смены направления движения головок пренебречь)?
    - 362 мс
    - 268 мс
    - 188 мс
101. Какие операционные системы позволяют взаимодействовать удаленным процессам и имеют сходное строение с автономными вычислительными системами?
    - сетевые операционные системы
    - распределенные операционные системы
    - операционные системы, поддерживающие работу многопроцессорных вычислительных систем
102. Сколько удаленных адресов может иметь сетевой компьютер?
    - только один
    - не более двух
    - потенциально произвольное количество
103. Пусть в некоторой сетевой операционной системе существует три различных протокола транспортного уровня, использующих собственные адресные пространства портов. Сколько типов сокетов существует в такой системе?
    - один
    - три
    - зависит от реализации
104. Пусть у нас есть локальная вычислительная сеть, достаточно долгое время работающая с неизменной топологией и без сбоев. Какие алгоритмы маршрутизации гарантируют доставку пакетов данных от отправителя к получателю по кратчайшему пути?
    - алгоритмы лавинной маршрутизации
    - алгоритмы состояния связей
    - маршрутизация от источника данных

105. Пусть у нас есть локальная вычислительная сеть, достаточно долгое время работающая с неизменной топологией и без сбоев. Какие алгоритмы маршрутизации гарантируют доставку пакетов данных по кратчайшему пути?
   - алгоритмы фиксированной маршрутизации
   - векторно-дистанционные алгоритмы с метрикой количества переходов между компонентами сети
   - алгоритмы случайной маршрутизации

106. Какие категории средств связи используются при взаимодействии удаленных процессов?
   - сигнальные
   - канальные
   - разделяемая память

107. Какой уровень эталонной модели OSI/ISO отвечает за создание контрольных точек при общении удаленных процессов?
   - сетевой уровень
   - транспортный уровень
   - уровень сеанса

108. Какой уровень эталонной модели OSI/ISO отвечает за доставку информации от компьютера-отправителя к компьютеру-получателю?
   - сетевой уровень
   - транспортный уровень
   - уровень сеанса

109. Какой уровень эталонной модели OSI/ISO отвечает за доставку информации от процесса-отправителя процессу-получателю?
   - сетевой уровень
   - транспортный уровень
   - уровень приложений

4. Примеры тем курсовых работ

Тема курсовой работы дается по усмотрению преподавателя (с учетом пожеланий студентов, которые могут предлагать свои темы), результат реализации должен включать взаимодействие с пользователем (пользовательский интерфейс в виде меню, желательно графического, хотя возможны и текстовые версии при условии использования ООП библиотек). Курсовая работа может выполняться коллективно (2–3 чел.), должна включать параллельное исполнение компонент (например, включать нити или несколько взаимодействующих процессов) и использовать примитивы синхронизации (обязательно). Размер курсовой – не менее 600 строк на C# или C++, написанных одним человеком. Также должны использоваться коллекции данных по выбору автора (АТД – *ATL*, *STL* и т.д.).

Тема курсовой работы утверждается не позже, чем в феврале.

Все картинки должны рисоваться в 4 цветах с использованием алгоритмов цветового размазывания (дизеринга).

1. **Дано:**

прямоугольный параллелепипед с данными на равномерной прямоугольной сетке (в каждой точке узла 3D сетки задано скалярное значение)

Построить: объемные изоповерхности заданной функции с освещением от фиксированного направления. Наблюдение проводится также с фиксированных направлений (26 - 6 торцов куба, 8 вершин под 45 гр., 12 ребер под углом 45 гр.)

в файле в текстовом виде задано:

N M K - число узлов по каждой оси (x, y, z) - типичное значение - 100 x 100 x 100

затем NxMxK чисел - значения функции в соответствующих узлах

Метод построения: непрозрачными кубиками стандартного вида. Кубик считается непрозрачным, если хотя бы одно угловое значение больше фиксированного значения изоповерхности.
Для каждого фиксированного направления экран подразделяется на "столбики", перпендикулярные к экрану. Проекция любого кубика в этом столбике будет одинаковой, т.е. если мы будем перебирать кубики начиная от ближайших к экрану, то найдя непрозрачный, мы можем не анализировать все оставшиеся в столбике и переходить к следующим столбикам. Это определяет следующий алгоритм:
1 в соответствие с углом зрения разбиваем все кубики на столбики, проецирующиеся на одинаковые площадки.
2 внутри каждого столбика сортируем кубики по расстоянию до экрана
3 начинаем перебор от экрана: как только мы нашли непрозрачный кубик (где хотя бы одно из 8 угловых значений больше нашего фиксированного значения изоповерхности), мы рисуем его на экране с учетом расстояния до него и угла зрения (пользуясь стандартными формулами яркости - см книгу Роджерса), и переходим к следующему столбику.
Задание: создать программу, позволяющую считывать разные файлы с исходными данными, выбирать точку зрения, вектор освещения, help. Каждая грань кубика рисуется однотонной, но цвет ее определяется в зависимости от ее положения в пространстве в соответствие с формулами расчета освещенности.
Исполнители: 1 - меню (выбор точек зрения), 1 - собственно алгоритм.

5. **Дано:**

прямоугольный параллелепипед с данными на равномерной прямоугольной сетке (в каждой точке узла 3D сетки задано скалярное значение)
Построить: объемные изоповерхности заданной функции с освещением от фиксированного направления.
в файле в текстовом виде задано:
N M K - число узлов по каждой оси (x, y, z) - типичное значение - 10 x 10 x 10
затем NxMxK чисел - значения функции в соответствующих узлах
Метод построения: явное нахождение точек изоповерхности решением уравнения.
Для каждого пиксела экрана проводится перпендикуляр (луч зрения), причем определяются математические координаты этого луча. Затем находится пересечение этого луча с изоповерхностью (решая уравнение F(x0,y0,z) = Const по z, и получая точку пересечения луча с изоповерхностью (x0,y0,z0) - подразумевается, что луч испущен из точки (x0,y0) и идет параллельно оси z, а корень этого уравнения выбирается ближайший к экрану). В полученном значении точки определяется нормаль к этой изоповерхности, и в учетом расстояния до точки и угла зрения (пользуясь стандартными формулами яркости - см книгу Роджерса) ставим точку на экран. Обратите внимание на выбор направления нормали (внешняя или внутренняя).
Для определения значения внутри ячейки использовать трилинейную интерполяцию, для определения точки пересечения - методы простой итерации, Ньютона, деления пополам. Для экономии расчетного времени можно искать решение не в каждой точке, а с некоторым шагом по горизонтали и вертикали, заполняя значения в промежутке с помощью билинейной интерполяции цветов.
Задание: создать программу, позволяющую считывать разные файлы с исходными данными, выбирать точку зрения, вектор освещения, метод расчета яркости, метод расчета корня уравнения, help.
Исполнители: 1 - меню (выбор точек зрения), 1 - алгоритм интерполяции, значений внутри ячеек и решения уравнений.

6. **Дано:**

прямоугольный параллелепипед с данными на равномерной прямоугольной сетке (в каждой точке узла 3D сетки задано скалярное значение)

Построить: объемные изоповерхности заданной функции с освещением от фиксированного направления, обозрение с фиксированного направления.
в файле в текстовом виде задано:
N M K - число узлов по каждой оси (x, y, z)
затем NxMxK чисел - значения функции в соответствующих узлах
Метод построения: разбиение поверхности на достаточно малые куски.
Вся математическая область считается погруженной в куб, разбитый на ячейки, число которых является степенью двойки. Внутри маленького кубика восполнение значения проводится трилинейной интерполяцией. Наблюдение всегда проводится с торца куба.
Алгоритм является рекурсивным (алгоритм Варнока). На каждом шаге рекурсии рассматривается куб. Если он простой (т.е. или он целиком прозрачный - внутри него нет соответствующих значений изоповерхности, или целиком может быть построен как единое целое - например, он полностью содержится в самом маленьком кубике сетки), то он отрисовывается целиком или игнорируется, в противном случае осуществляется разбиение его на 8 меньших кубиков (делением пополам по каждой оси), и процедура повторяется для каждого из кубиков. При повторении учитывается видимость, т.е. если ближний к экрану кубик целиком непрозрачен, то следующий за ним от экрана кубик не рассматривается. Изображение непрозрачной области кубика делается с использованием трилинейной интерполяции - участок изоповерхности представляется куском билинейной поверхности, которая затем изображается с учетом расстояния до точек экрана и угла зрения (пользуясь стандартными формулами яркости - см книгу Роджерса).
Задание: создать программу, позволяющую считывать разные файлы с исходными данными, выбирать точку зрения, вектор освещения, метод расчета яркости, метод расчета корня уравнения, help. Вариация задачи - добавить программу пересчета (переинтерполяции) на новую сетку данных при изменении угла зрения.
Исполнители: 1 - меню, 1 - алгоритм разбиений и определение видимости, 1 - отрисовка билинейных поверхностей (+1 на переинтерполяцию - если потребуется).

7. **Дано:**

Набор хаотически заданных точек в двумерном пространстве со скалярным значением в каждой из них
Построить: изолинии значений, произвольные сечения, а так же цветовую карту используя ограниченный набор цветов. Так же надо построить в трехмерном объеме карту высот (как в задаче номер 5)
в файле в текстовом виде задано:
N - число узлов
затем Nx3 чисел - координаты (x,y) и значения функции в соответствующих узлах
Метод построения: триангуляция полученного набора точек, а затем построение изолиний методом отслеживания пересечений (используя линейную интерполяцию значений внутри треугольников). Кроме того, осуществляется построение цветовой карты - линейной интерполяции цвета в соответствии со значениями в узлах и заданным шаблоном градаций.
Триангуляция набора точек - это разбиение набора точек на треугольники так, что этот набор треугольников покрывает выпуклую оболочку набора точек плотно, без пробелов и пересечений, кроме того, минимальный угол треугольника в этом наборе является максимальным. Триангуляция осуществляется следующим алгоритмом:
1 создается треугольник, заведомо вмещающий в себя все точки (возможно, углы этого треугольника могут быть созданы специально).
2 добавляется новая точка из набора. При этом определяется треугольник, в который попала эта точка, и к имеющемуся набору треугольников добавляется новые три. В конце этой процедуры удаляются все треугольники, углами которых является искусственно добавленные точки.

3 поле того, как мы получили набор плотно покрывающих треугольников, проведем оптимизацию полученного набора следующим методом: возьмем два треугольника, соприкасающихся одной стороной., и проверим, не находится ли вершина какого-либо треугольника внутри окружности, описанной вокруг другого треугольника. Если да, то проведем операцию под названием “флип”: удалим общую сторону этих треугольников и добавим вместо нее другую, соединяющую противоположные стороны получившегося четырехугольника. После этой операции поместим в стек все треугольники, граничащие с затронутыми данной операцией, и для всех них повторим тест на окружность. После того, как все треугольники оказываются проверенными, полученное разбиение будет удовлетворять критериям оптимальности.
На наборе треугольников отслеживаются точки пересечения изолиний со сторонами треугольников (пользуясь линейной интерполяцией.). Через эти точки проводятся (пользуясь квадратичной интерполяцией для улучшения внешнего вида) изолинии. Их желательно рисовать толщиной в 3 точки.
Должна иметься возможность выбора двух точек, через которые проводится сечение, вдоль которого должен быть построен график зависимости значения от расстояния.
Кроме того, каждый из треугольников может быть нарисован следующим образом: для минимального и максимального значения скаляра в узлах устанавливается некое значение (от 0.0 до 1.0), соответствующее значениям шкалы цветов, в простейшем случае - шкале серых цветов от белого до черного; затем осуществляется по-точечная растеризация треугольника с использованием алгоритма Брезенхейма, и каждой точке экрана сопоставляется плавающее число из указанного диапазона (пользуясь линейной интерполяцией по 3 углам треугольника). Затем осуществляется постановка точки на экран с использованием алгоритмов “размазывания” (dithering) - подробно они описаны в кн. Роджерса.
Задание: создать программу, позволяющую считывать разные файлы с исходными данными, осуществляющую триангуляцию, позволяющую выбирать метод размазывания цветов и построения изолиний, help.
Исполнители: 1 - меню + выбор, 1 - триангуляция, 1 - изолинии + график, 1 - рисование цветовой карты, 1 на трехмерную поверхность.

8. **Дано:**

данные на двумерной неравномерной прямоугольной сетке (в каждой точке узла 2D сетки задано скалярное значение)
Построить: “Горбы” - объемное поле высот заданной функции с освещением от фиксированного направления, обозрение с фиксированного направления.
в файле в текстовом виде задано:
N M - число узлов по каждой оси (x, y)
N и M чисел - координаты разбиений сетки по соответствующим осям
затем NxM чисел - значения функции в соответствующих узлах
Метод построения: сортировка прямоугольников сетки по глубине, затем изображение каждого прямоугольника (как 2-х треугольников) в проекции на экран с учетом освещения.
Все полученные треугольники сортируются по удаленности от экрана, отбрасываются те треугольники, нормаль к которым составляет тупой угол с направлением к экрану, затем для каждого угла треугольника определяется его цвет в зависимости от расстояния до экрана, угла зрения и нормали (пользуясь стандартными формулами яркости - см книгу Роджерса). Затем осуществляется отображение треугольников на экране с использованием заливок Фонга или Гуро.
Задание: создать программу, позволяющую считывать разные файлы с исходными данными, выбирать точку зрения, вектор освещения, метод расчета яркости и отображения help.
Исполнители: 1 - меню, 1 - отрисовка треугольников.

9. **Дано:**
Набор хаотически заданных точек в трехмерном пространстве со скалярным значением в каждой из них
Построить: трехмерную изоповерхность значений
в файле в текстовом виде задано:
N - число узлов
затем Nx4 чисел - координаты (x,y,z) и значения функции в соответствующих узлах
Метод построения: триангуляция (тетраэдеризация) полученного набора точек, а затем построение изоповерхностей методом отслеживания пересечений.
Тетраэдеризация набора точек - это разбиение набора точек на тетраэдры так, что этот набор тетраэдров покрывает выпуклую оболочку набора точек плотно, без пробелов и пересечений, кроме того, минимальный угол тетраэдра в этом наборе является максимальным. Тетраэдеризация осуществляется следующим алгоритмом:
1 создается тетраэдр, заведомо вмещающий в себя все точки (возможно, вершины этого тетраэдра могут быть созданы специально).
2 добавляется новая точка из набора. При этом определяется тетраэдр, в который попала эта точка, и к имеющемуся набору тетраэдров добавляется новые четыре (а старый удаляется). В конце этой процедуры удаляются все тетраэдры, вершинами которых является искусственно добавленные точки.
3 поле того, как мы получили набор плотно покрывающих тетраэдров, проведем оптимизацию полученного набора следующим методом: возьмем два тетраэдра, соприкасающихся одной гранью, и проверим, не находится ли вершина какого-либо тетраэдра внутри сферы, описанной вокруг другого тетраэдра. Если да, то проведем операцию под названием «флип»: удалим общую грань этих тетраэдров и добавим вместо нее другую, соединяющую противоположные стороны получившейся фигуры. После этой операции поместим в стек все тетраэдры, граничащие с затронутыми данной операцией, и для всех них повторим тест на сферу. После того, как все тетраэдры оказываются проверенными, полученное разбиение будет удовлетворять критериям оптимальности.
На наборе тетраэдров отслеживаются точки пересечения изоповерхности со сторонами тетраэдров (пользуясь линейной интерполяцией) - полученный набор треугольников затем визуализируется.
Каждый из треугольников полученных в результате вычисления пересечения изоповерхности с ребрами тетраэдра может быть нарисован следующим образом: для каждой вершины изоповерхности вычисляется усредненная нормаль (как среднее арифметическое нормалей примыкающих поверхностей) и цвет вершины; затем осуществляется по-точечная растеризация треугольника с использованием алгоритма Брезенхейма пользуясь линейной интерполяцией по 3 углам треугольника. Затем осуществляется постановка точки на экран с использованием алгоритмов “размазывания” (dithering) - подробно они описаны в кн. Роджерса.
Задание: создать программу, позволяющую считывать разные файлы с исходными данными, осуществляющую триангуляцию, позволяющую выбирать метод размазывания цветов и построения изолиний, help.
Исполнители: 1 - меню, 1-2 - триангуляция, 1 - отрисовка треугольников.

Каждому студенту выдается один вариант из 1-го задания и из 2-го задания. Тема курсового проекта согласовывается с преподавателем не позднее конца февраля. Защита проекта производится путем презентации проекта всеми участниками – с показом визуальной презентации и демонстрации работы программы. Оценка по результатам защиты выставляется с учетом оценок за 2 задания, которые должны быть сданы студентом не позднее чем за неделю до предзащиты проекта.

4. Критерии оценивания

За каждый контрольный вопрос из контрольного задания студент получает от 0 до максимального балла в зависимости от полноты представленного ответа (решения). Критерии проставления баллов утверждаются на заседании учебно-методической комиссии кафедры. Процент суммарно набранных баллов от максимально возможного количества определяет оценку за теоретические знания по каждому контрольному заданию:

Оценка	Набранные баллы
отлично (10)	более 88%
отлично (9)	от 78% до 88% включительно
отлично (8)	от 68% до 78% включительно
хорошо (7)	от 58% до 68% включительно
хорошо (6)	от 48% до 58% включительно
хорошо (5)	от 38% до 48% включительно
удовлетворительно (4)	от 28% до 38% включительно
удовлетворительно (3)	от 18% до 28% включительно
неудовлетворительно (2)	от 08% до 18% включительно
неудовлетворительно (1)	не более 08%

Каждая лабораторная работа и задача из задания при полном правильном решении оценивается в определенное количество баллов от 5 до 25. Баллы за полностью правильное решение определяются преподавателями и утверждаются на заседании учебно-методической комиссии кафедры. Процент суммарно набранных баллов от максимально возможного количества определяет оценку за практические знания студента по вышеприведенной таблице.

Итоговая оценка за дифференцированный зачет выставляется на основании оценок, полученных за контрольные работы по теории (далее x и y), и оценки за практические знания, (далее z) по следующей формуле (x+y)*z/(z+(x+y)/2), с округлением результата до целого сверху.

5. Методические материалы, определяющие процедуры оценивания знаний, умений, навыков и (или) опыта деятельности

Промежуточная аттестация по дисциплине «Теория и технология программирования» в 3 (осеннем) семестре осуществляется в форме дифференцированного зачета. Дифференцированный зачет выставляется по результатам работы студента в течение семестра на основе оценок за 17 лабораторных работ, 2 задания, 2 контрольные работы.

Промежуточная аттестация по дисциплине «Теория и технология программирования» в 4 (весеннем) семестре осуществляется в форме дифференцированного зачета. Дифференцированный зачет выставляется по результатам работы студента в течение семестра на основе оценок за 17 лабораторных работ, 2 задания, защиты курсовой работы.

Время проведения каждой контрольной работы составляет 2-4 академических часа.

Во время проведения контрольных работ обучающиеся могут пользоваться программой дисциплины и любыми материалами.

Тема курсовой работы дается по усмотрению преподавателя (с учетом пожеланий студентов, которые могут предлагать свои темы), результат реализации должен включать взаимодействие с пользователем (пользовательский интерфейс в виде меню, желательно графического, хотя возможны и текстовые версии при условии использования ООП библиотек). Курсовая работа может выполняться коллективно (2–3 чел.), должна включать параллельное исполнение компонент (например, включать нити или несколько взаимодействующих процессов) и использовать примитивы синхронизации (обязательно). Размер курсовой – не менее 600 строк на C# или C++, написанных одним человеком. Также должны использоваться коллекции данных по выбору автора (АТД – *ATL, STL* и т.д.).

Тема курсовой работы утверждается не позже, чем в феврале.