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

УТВЕРЖДЕНО
Директор физтех-школы радиотехники и компьютерных технологий
Д.А. Гаврилов

Рабочая программа дисциплины (модуля)

по дисциплине: Технологии программирования в операционных системах
по направлению: Информатика и вычислительная техника
профиль подготовки: Компьютерные технологии и вычислительная техника
Физтех-школа Радиотехники и Компьютерных Технологий
кафедра радиоэлектроники и прикладной информатики
курс: 2
квалификация: бакалавр

Семестр, формы промежуточной аттестации: 3 (осенний) - Дифференцированный зачет

Аудиторных часов: 60 всего, в том числе:
лекции: 0 час.
семинары: 0 час.
лабораторные занятия: 60 час.

Самостоятельная работа: 75 час.

Всего часов: 135, всего зач. ед.: 3

Количество контрольных работ, заданий: 4

Программу составил: И.Р. Дединский, старший преподаватель

Программа обсуждена на заседании кафедры радиоэлектроники и прикладной информатики 30.01.2024

Аннотация

В рамках данного курса студенты получат знания и навыки построения и функционирования современных операционных систем. Будут рассмотрены общие концепции построения систем на примере Unix, проблемы их безопасности, процессы и процедуры планирования в операционных системах. Будет рассмотрен ряд важных подсистем: файловая система, система управления вводом-выводом, система управления памятью. Обучающиеся получат представление и опыт параллельного программирования, работы с разделяемой памятью.

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
УК-1 Способен осуществлять поиск, критический анализ и синтез информации, применять системный подход для решения поставленных задач	УК-1.2 Находит, критически анализирует и выбирает информацию, необходимую для решения поставленной задачи
УК-4 Способен осуществлять деловую коммуникацию в устной и письменной формах на государственном языке Российской Федерации и иностранном(ых) языке(ах)	УК-4.2 Использует современные информационно-коммуникативные средства для коммуникации
ОПК-2 Способен использовать современные информационные технологии и программные средства при решении задач профессиональной деятельности, соблюдая требования информационной безопасности	ОПК-2.1 Способен применять современные вычислительную технику и сервисы сети Интернет в области (сфере) профессиональной деятельности
ОПК-3 Способен составлять и оформлять научные и (или) технические (технологические, инновационные) отчеты (публикации, проекты)	ОПК-3.3 Владеет методами визуального и графического представления результатов научной (научно-технической, инновационной технологической) деятельности в виде отчетов, научных публикаций
ПК-3 Способен формализовать и алгоритмизировать поставленную задачу, написать программный код с использованием языков программирования, оформить код в соответствии с установленными требованиями	ПК-3.2 Владеет методами и приемами формализации и алгоритмизации задач
ПК-6 Способен приобретать и использовать организационно-управленческие навыки в профессиональной и социальной деятельности	ПК-6.2 Способен модернизировать организацию рабочего процесса с учетом современных методик, негативного опыта и особенностей поставленных задач

ПК-7 Способен составлять и контролировать план выполняемой работы, планировать необходимые для выполнения работы ресурсы, оценивать результаты собственной работы	ПК-7.2 Владеет современными методиками и навыками работы с программным обеспечением по отслеживанию сроков выполнения плана работы коллектива

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;
- 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

		Трудоемкость по видам учебных занятий, включая самостоятельную работу, час.

№	Тема (раздел) дисциплины	Лекции	Семинары	Лаборат. работы	Самост. работа
1	Введение			4	7
2	Контрольная работа 1			4	7
3	Контрольная работа 2			2	7
4	Кооперация процессов			20	14
5	Проблемы безопасности операционных систем			2	7
6	Процессы и их планирование в операционной системе			8	7
7	Сети и сетевые операционные системы			4	7
8	Система управления вводом выводом			8	7
9	Управление памятью			4	7
10	Файловые системы			4	5
Итого часов				60	75
Подготовка к экзамену		0 час.			
Общая трудоёмкость		135 час., 3 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.

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

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

6.Перечень рекомендуемой литературы

Основная литература

1. Основы операционных систем, Электрон. версия печ. публикации / В. Е. Карпов, К. А. Коньков. — Москва, ИНТУИТ, 2016

Дополнительная литература

фонд литературы кафедры:

1. Стивенс У. UNIX: Взаимодействие процессов. – СПб.: Питер, 2002.
2. Стивенс У. UNIX: Разработка сетевых приложений. – 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 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 «Технологии программирования в операционных системах» 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 занятий, учебной и научной литературе), доказательство отдельных утверждений, свойств;

- решение задач, предлагаемых студентам на 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 преподавателю, проводящему лабораторные работы.

ПРИЛОЖЕНИЕ

ОЦЕНОЧНЫЕ МАТЕРИАЛЫ ПО ДИСЦИПЛИНЕ (МОДУЛЮ)

по направлению: Информатика и вычислительная техника
профиль подготовки: Компьютерные технологии и вычислительная техника
Физтех-школа Радиотехники и Компьютерных Технологий
кафедра радиоэлектроники и прикладной информатики
курс: 2
квалификация: бакалавр

Семестр, формы промежуточной аттестации: 3 (осенний) - Дифференцированный зачет

Разработчик: И.Р. Дединский, старший преподаватель

1. Компетенции, формируемые в процессе изучения дисциплины

Код и наименование компетенции	Индикаторы достижения компетенции
УК-1 Способен осуществлять поиск, критический анализ и синтез информации, применять системный подход для решения поставленных задач	УК-1.2 Находит, критически анализирует и выбирает информацию, необходимую для решения поставленной задачи
УК-4 Способен осуществлять деловую коммуникацию в устной и письменной формах на государственном языке Российской Федерации и иностранном(ых) языке(ах)	УК-4.2 Использует современные информационно-коммуникативные средства для коммуникации
ОПК-2 Способен использовать современные информационные технологии и программные средства при решении задач профессиональной деятельности, соблюдая требования информационной безопасности	ОПК-2.1 Способен применять современные вычислительную технику и сервисы сети Интернет в области (сфере) профессиональной деятельности
ОПК-3 Способен составлять и оформлять научные и (или) технические (технологические, инновационные) отчеты (публикации, проекты)	ОПК-3.3 Владеет методами визуального и графического представления результатов научной (научно-технической, инновационной технологической) деятельности в виде отчетов, научных публикаций
ПК-3 Способен формализовать и алгоритмизировать поставленную задачу, написать программный код с использованием языков программирования, оформить код в соответствии с установленными требованиями	ПК-3.2 Владеет методами и приемами формализации и алгоритмизации задач
ПК-6 Способен приобретать и использовать организационно-управленческие навыки в профессиональной и социальной деятельности	ПК-6.2 Способен модернизировать организацию рабочего процесса с учетом современных методик, негативного опыта и особенностей поставленных задач
ПК-7 Способен составлять и контролировать план выполняемой работы, планировать необходимые для выполнения работы ресурсы, оценивать результаты собственной работы	ПК-7.2 Владеет современными методиками и навыками работы с программным обеспечением по отслеживанию сроков выполнения плана работы коллектива

2. Показатели оценивания компетенций

В результате изучения дисциплины «Технологии программирования в операционных системах» 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;
- средствами использования стандартных библиотек.

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

Примеры заданий:

1. Прочитаем ELF-файл утилитой readlelf -l. Заметим, что на разные секции стоят разные настройки прав доступа: одни секции можно читать и писать, другие - читать и испол-нять, третьи - просто читать. Объясните, почему, и поясните по выводу утилиты, какие секции файла соответствуют каким секциям виртуальной памяти процесса.
2. В описании вызовов типа exec* в MAN указан код возврата только для случая, когда вы-зов завершился с ошибкой. Почему?
3. Дают ли вызовы read/write гарантию целостного вычитывания/записи всех приня-тых/готовых к приёму данных? Почему на практике работы с каналами (с любыми пото-ковыми средствами IPC) их предпочитают оборачивать в цикл?
4. Приведите вариант реализации подудуплексного и дуплексного потокового IPC для ОС Linux на ЯП C, пользуясь для этого IPC "канал" (pipe, FIFO).
5. Почему для передачи данных между процессами разделяемая память и канальные при-митивы IPC (pipe, FIFO) эффективнее, чем очереди сообщений?

6. Почему при выводе значения указателя на один и тот же глобальный объект (массив, пе-ременную, и пр.) при двух разных запусках одной той же программы, скомпилирован-ной с флагом -fno-PIE, мы наблюдаем разные значения? Какие значения мы будем наблюдать, если установим /proc/sys/kernel/randomize_va_space в 0? Зачем нужна эта пе-ременная?

7. Как посредством вызова clone создать поток, аналогичный потоку, создаваемому по-средством pthread_create по умолчанию?

8. Почему при долгом рассчёте вычислительных задач на потоках хорошей практикой яв-ляется "привязывание" (affinity setting) потоков к одним и тем же ядрам CPU?

9. Для каких сигналов гарантирован хотя бы порядок доставки для сигналов одного типа? За счёт чего?

10. Предложите и реализуйте способ передачи файла между двумя процессами путём отоб-ражения содержимого на разделяемую память.

Перечень типовых заданий:

1. Эмулятор терминала

Разработать эмулятор терминала, позволяющий:

- Запускать программы, циклически считывая команды через STDIN и запуская их в процессах-потомках.
- Анализировать коды возврата завершенных процессов
- Запускать конвейер процессов, например «env| grep HOSTNAME | wc». Постарайтесь при этом минимизировать количество открытых одновременно файловых дескрипторов.

2. Дуплексный PIPE для Linux

Разработать структуру, реализующую дуплексный канал межпроцессного взаимодействия на неименованных каналах (pipe). Использовать при этом отдельные принципы построения программ на C в ООП-стиле (вспомнить пример 1 с семинара). Постараться максимизировать производительность эхо-теста с передачей от процесса-родителя к потомку и обратно больших файлов (> 4Гб) по частям. Метрика производительности - суммарное время, затраченное на передачу данных. Чем меньше, тем лучше.

3. Эксперименты с IPC

Сравнить на практике 3 способа взаимодействия между двумя процессами.

Для этого:

1) Разработать 3 программы, передающие файл произвольного (довольно большого) размера между двумя процессами с использованием следующих инструментов:

а) Разделяемая память SYS V.

б) Очереди сообщений SYS V.

в) Именованые каналы (FIFO).

2) Для каждой из программы измерить время выполнения при передаче файла, используя утилиту time либо средства из заголовка time.h: https://ru.wikipedia.org/wiki/Time.h, проводя измерения для 2-3 характерных размеров буфера (памяти, данных в сообщении): "малый" размер - менее пары сотен байт, 4 Кб, большой размер - существенно больше 4 Кб.

3) По результатам измерений построить 3 гистограмы времени передачи от размера буфера. На каждой гистограмме будут расположены 3 полосы, каждая для инстумента 1а, 1б, 1в соответственно. Попробовать поймать ситуации, в которых результаты будут наиболее репрезентативными. После чего следует сделать выводы и заключения.

4. Распределённый рассчёт интеграла методом Монте-Карло

Рассчитать определённый интеграл какой-нибудь простой, интегрируемой аналитически на некотором промежутке функции f(x) методом Монте-Карло:

расчёт проводить параллельно в n-потоков некоторой программы А, генерируя в каждом из них N/n точек равномерно как по интервалу, так и по области значений.

Собирать и обрабатывать результаты в программе Б, читающей результаты из разделяемой памяти. [В случае использования независимых ячеек разделяемой памяти,

сохранение результатов можно обеспечить без синхронизации, в случае разделяемой ячейки в разделяемой памяти - синхронизация необходима. Реализация способа хранения
влияет на результаты эксперимента 2 а,б*) (см. ниже).

Задачи, поставленные перед Вами как исследователем:

1) Оценить прирост производительности в связи с использованием n' потоков вместо 1, где n' - число физических ядер Вашей машины.

Узнать число ядер-например, 'cat /cpu/procinfo'

2) Провести серию измерений, в которой, увеличивая число n' :

а) пронаблюдать стагнацию роста производительности при увеличении числа потоков

б*) определить, при каких n' (насколько больших) накладные расходы от использования многопоточности [и синхронизации, если она имела место]
превосходят преимущества их использования.

5. Передача файла сигналами

Передайте файл между двумя процессами с помошью сигналов, наиболее быстрым из известных Вам способом.

6. Фоновая служба

Разработать программу-демон, которая:

1) Находит процесс по его PID, читает его рабочую директорию и запись на procfs /proc/<pid name>

2) Выполняет следующее:

Вариант 1: рекурсивно просматривает рабочую директорию, находит в ней изменения только в текстовых файлах (см. пример на diff + подумайте,
как программно определить, что файл текстовый, используя утилиту file), после чего копирует изменённые файлы в другую директорию, то есть делает бекап.

Вариант 2: мониторит отображения виртуальной памяти, записывая изменения, например, отображение новых разделяемых библиотек, после чего сообщает в лог или какой-либо буфер.

Потенциальный плюсик в карму за инкрементальные бекапы (сохраняем только последовательности diff-ов и временную отметку создания,
потом их последовательно накатываем на исходные состояния файлов)

В любом из вариантов, демон должен сохранять данные в виде периодических семплов, то есть делать одну запись в T мс, где T - период семплирования, задаваемый
демону через некоторый канал IPC взаимодействия с ним (например, FIFO).

Вариант 3. Избранные поля-счетчики /proc/vmstat. При этом результаты нужно выводить как разность между показаниями семплов i-1 и i на i-м шаге.

Демон должен быть отвязан от управляющего терминала и быть защищённым от нежелательных сигналов и каналов ввода/вывода.

На приёме также будет производиться контроль утечки ресурсов.

7. FIFO-клиент-сервер

7-я домашка посвящена мультиплексированию ввода-вывода.

Предлагается разработать программное решение, реализующее в себе архитектуру "клиент-сервер". Данный тип архитектуры часто используется в организации сетевого взаимодействия программ. В виду того, что сеть мы ещё не прошли, в нашем примере взаимодействие организуется в рамках одного компьютера, и компоненты общаются между собой не средствами TCP/IP стека Linux, а через FIFO.

Программа делится на 2 части:

1) Клиент (Cn) - отправляет серверу запрос на регистрацию себя на нём, после чего имеет возможность скачивать файлы через сервер

2) Сервер (S) - регистрирует клиент, после чего отдаёт соответствующий файл по запросу

Рассмотрим схему:

1.Регистрация клиента происходит отправкой сообщения через специальное командное IPC (это может быть FIFO или очередь сообщений).

Команда регистрации содержит 3 токена: REGISTER <fifo/tx/path> <fifo/rx/path> - команда и пути к fifo-записям на ФС для передачи команд на скачивание / скачивания файлов соответственно. Заметим, что клиентов может быть больше одного (но немного, давайте до 64-х).

Сервер отвечает "ACK" клиенту по обратному каналу связи выбранного IPC, сохраняет данные и открывает 2 FIFO для взаимодействия с ним.

! FIFO на сервере следует открывать в O_RDWR-режиме (подробнее здесь: https://stackoverflow.com/questions/14594508/fifo-pipe-is-always-readable-in-select)

CMD INTERFACE:

REGISTER <file/tx/path> <fifo/rx/path>

(Cx) ----------> (S): open(fromclient_tx_fifo_fd, O_RDWR), open(fromclient_rx_fifo_fd, O_RDWR), then:

<----------ACK----------

2. Далее любой подключённый клиент может запросить у сервера файл командой GET <filename> по своему TX, сервер обязан определить, какой клиент это сделал, и отправить его по клиентскому RX-каналу. Список файлов известен клиенту и серверу, так что можно забить статически.

REQUEST-RESPONSE WORKFLOW:

GET <filname> (S):

(Cx) ----------> |fromclient_tx_fifo_fd|

<----------file---------- |fromclient_rx_fifo_fd|

Замечание по устройству сервера: сервер отслеживает состояние множества входящих дескрипторов через вызов select(...), определяет дескриптор, на который пришёл запрос, после чего запускает процедуру отдачи файла В ОТДЕЛЬНОМ ПОТОКЕ. Таким образом, при большом (в наших ограничениях :)) числе одновременных обращений сервер является неблокирующим. Пример использования select(...) см. в "examples/".

REQUEST-RESPONSE in more detail...:

(S):

(C1) |fromclient_tx_fifo_fd1| - \

|fromclient_rx_fifo_fd1|. \

... request is here! \

(Cx) ----------**********---------> |fromclient_tx_fifo_fdx| - select() --> FD_ISSET(x, &read_descr_set)==TRUE --> then-->|

<----------file---------- |fromclient_rx_fifo_fdx|.

<----------create_thread,get_file,transmit file|

..... /

(Cn) |fromclient_tx_fifo_fdn| /

(*)Задача со звёздочкой (потенциальный "плюс балл"): попробовать "переиспользовать" потоки для новых запросов/использовать "пул" подготовленных заранее потоков. NB. По данному пункту буду спрашивать жестко. В дополнение, могу сослаться на код (естественно, при слаче могу заинтересоваться внутренним устройством. И не гарантирую, что он везде рабочий...): https://github.com/Pithikos/C-Thread-Pool

8. Распределённый расчёт интеграла методом Монте-Карло с распараллеливанием по сети

Базовый уровень. разработать комплекс программ с архитектурой клиент-сервер, который запускается на нескольких компьютерах, причём среди запущенных программ есть:

Клиент, который отправляет задание посчитать часть задачи, а затем собирает результаты

Серверы, которые принимают от клиента задание в формате, специфицируемом разработчиком (число точек, интервал и т.д) через TCP-соединение, считают и отправляют результат обратно клиенту.

Список адресов серверов статически хранится у клиента и известен ему. Номера портов также известны и фиксированы. Клиент может периодически опрашивать сервер, запрашивая результат вычислений.

Повышенный уровень:

*(Плюс балл). Список адресов неизвестен клиенту, и определяется динамически: На серверах, готовых вычислить часть задачи, запущен UDP-сервер, отвечающий клиенту на некотором заранее известном порте.

Клиент рассылает UDP-широковещательное сообщение (BROADCAST) компьютерам в своей подсети. См. пример: http://beej.us/guide/bgnet/html/single/bgnet.html#broadcast

Ответы от клиентов записываются в специальный список адресов, по которым потом следует установить TCP-соединение и дать задание на расчёт.

*(Плюс балл). Балансировка нагрузки на серверах (статическая, без миграций подзадач): определяется число ядер на серверах, после чего клиент рассылает задания пропорционально этому числу. Например, в сети есть 2 вычислителя, 4 и 8 ядрами соответственно. Тогда исходная задача будет разделена в соотношении 1:2, и первая часть будет отдана компьютеру №1, а вторая, вдвое большая - компьютеру №2.

Ожидаемое поведение Вашего массово-параллельного решателя вне зависимости от условий выше - линейное ускорение от масштабирования по сети. В решении рекомендуется максимально переиспользовать код из task_8/examples.

За каждое задание выставляется отценка в соответствии со следующими критериями оценивания:

Отлично

10) Полностью и вовремя решены все задачи без ошибок. Продемонстрирован грамотный подход к решению задач, реализованы оптимальные алгоритмы, код оформлен в едином удобочитаемом стиле.
9) Полностью и вовремя решены все задачи без ошибок. Продемонстрирован грамотный подход к решению задач, реализованы оптимальные алгоритмы.
8) Полностью и вовремя решены все задачи без ошибок. Продемонстрирован грамотный подход к решению задач.

Хорошо

7) Полностью решены все задачи. Допущены несущественные ошибки

6) Полностью решено большинство задач. В некоторых задачах допущены и не исправлены ошибки, либо некоторые задачи решены частично.

5) Полностью решено две трети задач. В некоторых задачах допущены и не исправлены ошибки, либо некоторые задачи решены частично.

Удовлетворительно.

4) Полностью решено более половины задач. В остальных задачах допущены и не исправлены ошибки, либо некоторые задачи решены частично.

3) Полностью решено более половины задач.

Неудовлетворительно

2) Решено менее половины задач.

1) Не решено ни одной задачи.

4. Перечень типовых (примерных) вопросов и тем для проведения промежуточной аттестации обучающихся

1. Прочитаем ELF-файл утилитой readlelf -l. Заметим, что на разные секции стоят разные настройки прав доступа: одни секции можно читать и писать, другие - читать и испол-нять, третьи - просто читать. Объясните, почему, и поясните по выводу утилиты, какие секции файла соответствуют каким секциям виртуальной памяти процесса.

2. В описании вызовов типа exec* в MAN указан код возврата только для случая, когда вы-зов завершился с ошибкой. Почему?

3. Дают ли вызовы read/write гарантию целостного вычитывания/записи всех приня-тых/готовых к приёму данных? Почему на практике работы с каналами (с любыми пото-ковыми средствами IPC) их предпочитают оборачивать в цикл?

4. Приведите вариант реализации подудуплексного и дуплексного потокового IPC для ОС Linux на ЯП C, пользуясь для этого IPC "канал" (pipe, FIFO).

5. Почему для передачи данных между процессами разделяемая память и канальные при-митивы IPC (pipe, FIFO) эффективнее, чем очереди сообщений?

6. Почему при выводе значения указателя на один и тот же глобальный объект (массив, пе-ременную, и пр.) при двух разных запусках одной той же программы, скомпилирован-ной с флагом -fno-PIE, мы наблюдаем разные значения? Какие значения мы будем наблюдать, если установим /proc/sys/kernel/randomize_va_space в 0? Зачем нужна эта пе-ременная?

7. Как посредством вызова clone создать поток, аналогичный потоку, создаваемому по-средством pthread_create по умолчанию?

8. Почему при долгом рассчёте вычислительных задач на потоках хорошей практикой яв-ляется "привязывание" (affinity setting) потоков к одним и тем же ядрам CPU?

9. Для каких сигналов гарантирован хотя бы порядок доставки для сигналов одного типа? За счёт чего?

10. Предложите и реализуйте способ передачи файла между двумя процессами путём отоб-ражения содержимого на разделяемую память.

11. Почему при разработке фоновых служб (daemons), после создания процесса под службу, хорошей практикой является установка корневой директории как текущей?

12. Объясните, как пространства имён процессов (PID namespaces) и вызов prctl(PR_SET_CHILD_SUBREAPER, ...), лежащие в основе технологии контейнерной виртуализации на пространствах имён в Linux, обеспечивают изоляцию поддеревьев в дереве процессов.

13. Чем сценарий обычного монтирования вызовом mount отличается от bind mount? При-ведите пример.

14. Перечислите основные преимущества и недостатки вызовов select, poll, epoll. В чём за-ключается улучшение ppoll/pselect в сравнении с poll/select?

15. Назовите основные преимущества обработки установленных TCP-соединений в отдель-ных потоках.

Критерии оценивания

Оценка "отлично (10)" выставляется студенту, показавшему всесторонние, систематизированные, глубокие знания учебной программы дисциплины, проявляющему интерес к данной предметной области, продемонстрировавшему умение уверенно и творчески применять их на практике при решении конкретных задач, свободное и правильное обоснование принятых решений.

Оценка "отлично (9)" выставляется студенту, показавшему всесторонние, систематизированные, глубокие знания учебной программы дисциплины и умение уверенно применять их на практике при решении конкретных задач, свободное и правильное обоснование принятых решений.

Оценка "отлично (8)" выставляется студенту, показавшему всесторонние, систематизированные, глубокие знания учебной программы дисциплины и умение уверенно применять их на практике при решении конкретных задач, правильное обоснование принятых решений, с некоторыми недочетами.

Оценка "хорошо (7)" выставляется студенту, если он твердо знает материал, грамотно и по существу излагает его, умеет применять полученные знания на практике, но недостаточно грамотно обосновывает полученные результаты.

Оценка "хорошо (6)" выставляется студенту, если он твердо знает материал, грамотно и по существу излагает его, умеет применять полученные знания на практике, но допускает в ответе или в решении задач некоторые неточности.

Оценка "хорошо (5)" выставляется студенту, если он в основном знает материал, грамотно и по существу излагает его, умеет применять полученные знания на практике, но допускает в ответе или в решении задач достаточно большое количество неточностей.

Оценка "удовлетворительно (4)" выставляется студенту, показавшему фрагментарный, разрозненный характер знаний, недостаточно правильные формулировки базовых понятий, нарушения логической последовательности в изложении программного материала, но при этом он освоил основные разделы учебной программы, необходимые для дальнейшего обучения, и может применять полученные знания по образцу в стандартной ситуации.

Оценка "удовлетворительно (3)" выставляется студенту, показавшему фрагментарный, разрозненный характер знаний, допускающему ошибки в формулировках базовых понятий, нарушения логической последовательности в изложении программного материала, слабо владеет основными разделами учебной программы, необходимыми для дальнейшего обучения и с трудом применяет полученные знания даже в стандартной ситуации.

Оценка "неудовлетворительно (2)" выставляется студенту, который не знает большей части основного содержания учебной программы дисциплины, допускает грубые ошибки в формулировках основных принципов и не умеет использовать полученные знания при решении типовых задач.

Оценка "неудовлетворительно (1)" выставляется студенту, который не знает основного содержания учебной программы дисциплины, допускает грубейшие ошибки в формулировках базовых понятий дисциплины и вообще не имеет навыков решения типовых практических задач.

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

Дифференцированный зачет проводится по итогам текущей успеваемости и сдачи заданий, предусмотренных программой дисциплины.

При проведении дифференцированного зачета обучающемуся предоставляется 20 минут на подготовку. Опрос обучающегося проводится в течение 30 минут.

Во время проведения дифференцированного зачета обучающиеся могут пользоваться программой дисциплины.