ОПЕРАЦИОННЫЕ СИСТЕМЫ И СИСТЕМНОЕ ПРОГРАММИРОВАНИЕ

Лекция № 00 — Установочная лекция

Преподаватель: Поденок Леонид Петрович, 505а-5

+375 17 293 8039 (505а-5)
+375 17 320 7402 (ОИПИ НАНБ)
prep@lsi.bas-net.by

ftp://student:2ok*uK2@Rwox@lsi.bas-net.by

Кафедра ЭВМ, 2024

2024.02.09

Оглавление

Предмет курса, цели и задачи

Лекции	32
Лабораторные	36
Самостоятельная работа	52

Цель дисциплины

- изучение организации и принципов построения современных операционных систем, основанных на открытых международных стандартах;

- изучение использования аппаратных и программных средств современных процессоров, предназначенных для поддержки многозадачных операционных систем;

- изучение методологии разработки системно-ориентированных программ (утилит) с использованием современных алгоритмических языков и систем программирования.

Задачи учебной дисциплины

- формирование базовых знаний в области организации и функционирования современных операционных систем;

- изучение способов разработки системного программного обеспечения с учетом особенностей современных операционных систем;

- овладение методами разработки, тестирования, отладки и документирования программ, направленных на решение системных задач, с использованием современных инструментальных средств.

Должен знать

- принципы построения операционных систем;
- типовые алгоритмы организации взаимодействия между процессами;
- задачи, решаемые при управлении виртуальной памятью;
- основные системные вызовы стандарта POSIX;
- систему прав доступа в файловых системах ОС UNIX/Linux;
- способы взаимодействия процессов в ОС UNIX/Linux;
- способы межпрограммного взаимодействия с использованием сокетов;
- способы потокового взаимодействия в рамках одного процесса на уровне поддержки стандарта ISO/IEC 14882 2011+ (Язык программирования C++);
- способы потокового взаимодействия POSIX;
- архитектуру и подсистемы ОС UNIX/Linux;
- модель виртуальной памяти процесса;
- понятие совместно используемых объектов в ОС UNIX/Linux;
- средства поддержки многозадачности в ОС UNIX/Linux и методы синхронизации задач.

Должен уметь

- разрабатывать кроссплатформенные программы в ОС Linux;
- разрабатывать программы, по организации взаимодействия между процессами в рамках стандарта POSIX;
- разрабатывать программы, по организации потокового взаимодействия в рамках стандарта POSIX;
- разрабатывать многопоточные программы с синхронизацией данных в рамках стандарта POSIX;
- разрабатывать совместно используемые объекты (динамически подключаемые библиотеки);
- разрабатывать протоколы и способы межпрограммного взаимодействия;
- использовать проецируемые в память файлы в рамках стандарта POSIX.

Должен владеть

- современными технологиями проектирования системного программного обеспечения;
- современными технологиями тестирования, отладки, верификации, аттестации и оценки качества системного программного обеспечения;
- методами эффективной эксплуатации системного программного обеспечения.

Пререквизиты

№ п/п	Название дисциплины	Раздел, темы
1	Программирование на языках высокого уровня	Языки программирования C и C++
2	Программирование на языке ассемблера	Все
3	Основы алгоритмизации и программирования	Все

№	Наименование разделов, тем	Содержание тем
Основы операционных систем		
1	Понятия операцион-ной системы	Операционная система. Классификация ОС. ОС реального времени. Микроядерные и монолитные ОС. Структура ОС. Ядро, командный процессор, подсистема ввода-вывода, система управления памятью, файловая система. Понятие системных вызовов. Системные вызовы стандарта POSIX. Концепция виртуализации.

№	Наименование разделов, тем	Содержание тем
2	Понятия процесса и потока. Механизмы взаимного исключения	Концепция процесса. Диаграмма состояний процесса. Операции над процессами. Создание и завершение процесса. Иерархия процессов. Структуры управления процессами. Процессы зомби. Реализация процессов в современных ОС. Процессы и потоки. Понятия мультизадачности и многопоточности. Потоки в пространстве пользователя и потоки в ядре. Понятие о прерываниях. Параллельные процессы. Независимые и взаимодействующие процессы. Механизмы уведомления процессов о системных событиях. Взаимодействие процессов. Состояние состязания. Детерминированный набор и условия Бернстайна. Понятие критического ресурса. Критическая секция. Взаимное исключение. Механизмы взаимого исключения. Алгоритмы Деккера, Петерсона, Лэмпорта.
3	Типовые механизмы синхронизации	Операция Test & Set (TS). Поддержка механизма TS в современных процессорах. Семафоры. Базовые операции над семафорами. Мьютексы. Задача «поставщик-потребитель». Инверсия приоритетов. Механизмы синхронизации в современных ОС. Мониторы в языках программирования. Барьеры. Задача «читатели-писатели».

№	Наименование разделов, тем	Содержание тем
4	Ресурсы. Управление памятью. Организация виртуальной памяти.	Распределение ресурсов, проблема тупиков. Алгоритмы обнаружения тупиков и выхода из них. Требования к управлению памятью. Схемы распределения памяти. Страничная организация памяти. Управление виртуальной памятью – размещение, перемещение, преобразование адресов, замещение. Память процесса. Управление памятью в современных ОС. Защита памяти.
5	Планирование в операционных системах	Стратегии планирования. Дисциплины диспетчеризации. Вытесняющие и невытесняющие алгоритмы. Алгоритмы планирования без переключений. Циклическое и приоритетное планирование. Динамические приоритеты. Планирование в системах реального времени. Планирование потоков. Гарантии обслуживания процесса.

№	Наименование разделов, тем	Содержание тем
6	Управление вводом-выводом и файлами	Организация функций ввода-вывода. Логическая структура функций ввода-вывода. Буферизация операций ввода-вывода. Дисковое планирование. Система управления файлами. Организация файлов, доступ к файлам. Блочные и символьные операции. Синхронные и асинхронные операции. Отображение ввода–вывода на адресное пространство памяти. Прямой доступ к памяти. Кэширование операций. Упреждающее чтение. Отложенная запись. Программное обеспечение ввода–вывода. Организация драйверов в современных ОС. Псевдоустройства. Файлы устройства.
POSIX-совместимые операционные системы		
7	Введение	Основные принципы организации и построения ОС UNIX. Стандарты SUS и POSIX. Пользователи системы, атрибуты пользователя.
8	Основные утилиты	Системная оболочка. Структура каталогов. Работа с файлами. Управление пользователями и правами. Установка и управление программным обеспечением.
9	Файловая система	Структура файловой системы. Жесткие и символические ссылки. Работа с каталогами, чтение и запись файлов, управление файлами, управление устройствами.

№	Наименование разделов, тем	Содержание тем
10	Процессы в ОС класса UNIX	Атрибуты процессов. Создание процессов и управление ими. Завершение процессов. Синхронизация процессов. Запуск программы. Средства для управления свойствами процессов. Группы и сеансы.
11	Взаимодействие между процессами I	Сигналы. Обработка сигналов. Неименованные каналы. Именованные каналы. Сообщения, семафоры, совместно используемая память.
12	Потоки POSIX	Атрибуты потоков. Создание и управление потоками. Завершение потоков. Синхронизация потоков. Мьютексы. Переменные состояния. Спинлоки. RW-блокировки.
13	Эффективная работа с файлами	Файлы, отображаемые на память. Неблокирующий и асинхронный ввод-вывод.
14	Системное программирование	Связь ОС UNIX и языка С. Жизненный путь программ. Динамическое связывание. Структура объектного модуля. Системные вызовы и функции стандартных библиотек. Обработка ошибок.
15	Взаимодействие между процессами II	Сокеты. Типы, адреса. Принцип действия и порядок работы. Сокеты AF_LOCAL для локального межпроцессного взаимодействия.
16	Взаимодействие между процессами в сетевой среде	Сокеты AF_INET, AF_INET6, AF_PACKET

Литература

Основная

1) **Таненбаум, Э. Современные операционные системы** / Э. Таненбаум, Х. Бос. – 4-е изд. – Санкт-Петербург : Питер, 2021. – 1120 с. : ил.

2) **Стивенс, У. Р. UNIX. Профессиональное программирование** / У. Р. Стивенс, С. А. Раго ; пер. А. Киселева. – 2-е изд. – Санкт-Петербург : Символ-плюс, 2007. – 1040 с. : ил.

3) **Рочкинд, М. Д. Программирование для UNIX** / М. Д. Рочкинд. – 2-е изд., перераб. и доп. – Санкт-Петербург : Русская Редакция : БХВ-Петербург, 2005. – 704 с.

4) Гласс, Г. UNIX для программистов и пользователей / Г. Гласс, К. Эйблс. – 3-е изд., перераб. и доп. – Санкт-Петербург : БХВ-Петербург, 2004. – 848 с. : ил.

5) **Реймонд, Э. С. Искусство программирования для Unix** / Э. С. Реймонд ; пер. с англ. - Москва : Вильямс, 2005. – 544 с. : ил.

6) **Столяров, А. В. Программирование: введение в профессию** : в 3 т. Т. II : Системы и сети / А. В. Столяров. – 2-е изд., испр. и доп. – Москва : МАКС Пресс, 2021. – 704 с. : ил.

7) Робачевский, А. М. Операционная система UNIX : учебное пособие [доп. МО РФ] / А. М. Робачевский, С. А. Немнюгин, О. Стестик. – 2-е изд. – Санкт-Петербург : БХВ-Петербург, 2007. – 656 с. : ил.

8) **Стивенс, У. Р. UNIX : разработка сетевых приложений** / У. Р. Стивенс, Б. Феннер, Э. М. Рудофф. – 3-е изд. – Санкт-Петербург : Питер, 2007. – 1039 с. : ил.

Дополнительная

1) ISO/IEC 9899-2011[2012] – Programming languages – C, 702 p.

2) ISO/IEC 14882-2017[2017] – Programming languages – C++, – 1622 p.

3) IEEE Std 1003.1 -2017 (Revision of IEEE Std 1003.1-2008). IEEE Standard for Information Technology. Portable Operating System Interface (POSIX®). Base Specifications, Issue 7, 2017. – 3951 p.

4) Лав, Р. Ядро Linux. Описание процесса разработки / Р. Лав. – 3-е изд. – Москва : Вильямс, 2013. – 496 с. : ил.

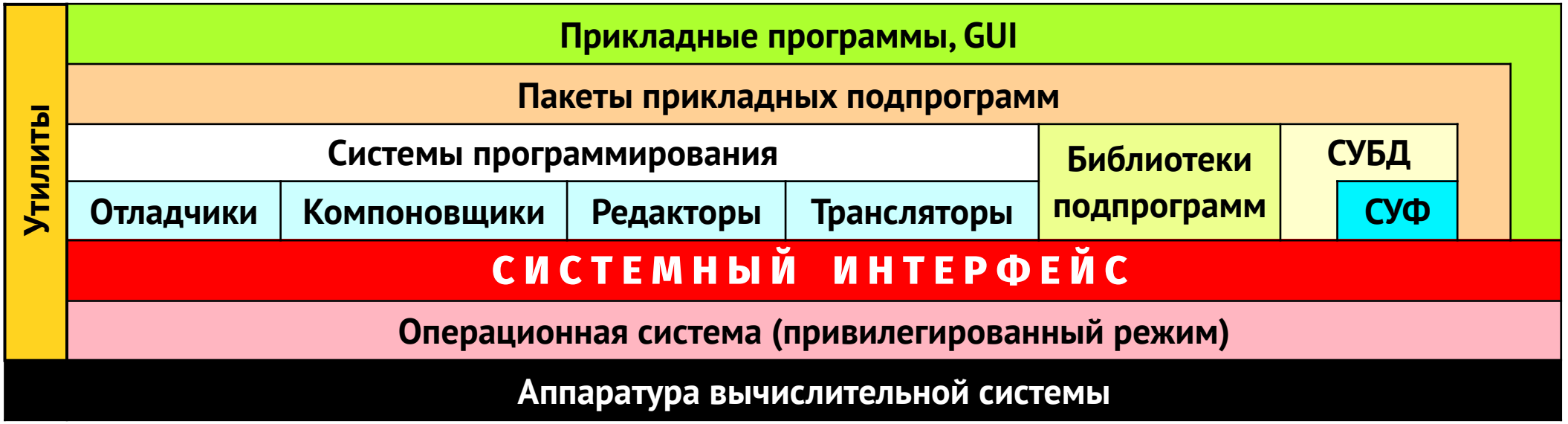
5) ГОСТ 19 Система программной документации (серия стандартов).

6) ГОСТ 2 Система конструкторской документации (серия стандартов).

http://libgen.rs — Library Genesis
http://rutracker.org
http://opennet.ru

Уровень рассмотрения

Обобщенная структура вычислительной системы



Наш курс, в основном, имеет отношение к слоям, которые на схеме обозначены как «СИСТЕМНЫЙ ИНТЕРФЕЙС» и «**Операционная система**»

Платформа и инструментарий

Операционная система — **Linux**. (базовый, Dual-boot, VM)

Рекомендуется: дистрибутивы на базе RPM (Fedora, RockyLinux, CentOS Stream).

Не рекомендуется: дистрибутивы на базе DEB (Ubuntu, …).

Десктоп

Рекомендуется: XFCE4, KDE

Не рекомендуется: GNOME

Ассемблер — **nasm**;

Компилятор C, C++ — **GNU Compiler Collection (gcc)** версия 10 и выше;

Сборка программ — **make**;

Отладчик для C — **gdb**;

Отладчик для nasm — **edb**;

Управление файлами — **mc** (Midnight Commander).

Компоновщик — **ld** (**binutils**);

Редактор текста или IDE, поддерживающие подсветку синтаксиса (**slickedit.com**).

Файл **stud_io.inc** из архива материалов к книге «Столяров А.В. Программирование: Введение в профессию. Т.2. Низкоуровневое программирование. 2016.pdf»

На чем писать код?

ISO/IEC 9899-2011 — C Programming language.

Ассемблер или прямо в машинных кодах

Лабораторные работы

№	Наименование лабораторной	Содержание
1.	Создание процессов и управление ими	Освоение командной строки оболочки. Базовая работа с процессами средствами оболочки bash.
2.	Взаимодействие и синхронизация процессов	Сигналы, семафоры, каналы, совместно используемая память.
3.	Создание потоков и управление ими	Базовая работа с потоками. Порождение, взаимодействие, завершение, очистка.
4.	Взаимодействие и синхронизация потоков	Сигналы, семафоры, каналы. Совместно используемая память.
5.	Файловая система	Работа с каталогами. Разработка простого аналога утилиты find
6.	Операции с файлами	Работа многопоточного приложения с файлами, отображенными на память, барьерная синхронизация.
7.	Взаимодействие процессов. Локальные сокеты.	Кооперация процессов для обработки данных
8.	Взаимодействие процессов в сетевой среде.	Реализация системы типа «клиент-сервер» с использованием сокетов

№ 1. Знакомство с Linux/Unix и средой программирования. POSIX-совместимая файловая система.

Внешнее знакомство с POSIX-совместимой файловой системой – структура каталогов, жесткие и символические ссылки, права доступа, монтирование файловых систем, монтирование каталогов (**mount**, **mount --bind**).

Команды и утилиты оболочки **man**, **info**, **mkdir**, **touch**, **rm**, **rmdir**, **cd**, **cat**, **sort**, **head**, **tail**, **tee**, **wc**, **chmod**, **ls**, **lsof**, **lsblk**, **lsusb**, **lscpu**, **ln**, **link**, **unlink**, **locale**, **iconv**, **kill**, **top**, **htop**, **ps**, **grep**, **diff**, **env**, **file**, **stat**, **find**, **tar**, **gzip**, **more**, **less**, **printf**, **time**, **...**

Сцепление программ и соединение выходных и входных стандартных потоков.

Перенаправление вывода **stdout** и **stderr** в файлы

Экосистема курса – **gcc**, **make**, **gdb**,

Структура ФС, содержимое **inode**, команды оболочки.

Знакомство с POSIX-совместимой файловой системой – **opendir(3)**, **readdir(3)**, **closedir(3)**, **fstat(2)**, **readlink(2)**, **realpath(1)**, **symlink(2)**, **link(2)**, **unlink(2)**, **...**

Задание

Освоить эффективную работу с файлами в оболочке и **mc**.

Разработать программу **dirwalk**, сканирующую файловую систему и выводящую в **stdout** информацию в соответствии с опциями программы.

Формат вывода аналогичен формату вывода утилиты **find**.

```
dirwalk [dir] [options]
```

dir — начальный каталог. Если опущен, текущий (**./**).

options — опции.

-l – только символические ссылки (**-type l**)

-d – только каталоги (**-type d**)

-f -- только файлы (**-type f**)

-s — сортировать выход в соответствии с **LC_COLLATE**

Опции могут быть указаны как перед каталогом, так и после.

Опции могут быть указаны как раздельно, так и вместе (**-l -d**, **-ld**).

Если опции **ldf** опущены, выводятся каталоги, файлы и ссылки.

Для обработки опций рекомендуется использовать **getopt(3)** или **gengetopt(1)**.

№ 2. Понятие процессов.

Изучение системных вызовов **fork(), execve(), getpid(), getppid(), getenv()**.

Задание

Разработать две программы – **parent** и **child**.

Перед запуском программы **parent** в окружении создается переменная среды **CHILD_PATH** с именем каталога, где находится программа **child**.

Родительский процесс (программа **parent**) после запуска получает переменные среды, сортирует их в **LC_COLLATE=C** и выводит в **stdout**. После этого входит в цикл обработки нажатий клавиатуры.

1) Символ «**+**», используя **fork(2)** и **execve(2)** порождает дочерний процесс и запускает в нем очередной экземпляр программы **child**. Информацию о каталоге, где размещается **child**, получает из окружения, используя функцию **getenv()**. Имя программы (**argv[0]**) устанавливается как **child_XX**, где XX – порядковый номер от 00 до 99. Номер инкрементируется родителем.

2) Символ «*» порождает дочерний процесс аналогично предыдущему случаю, однако информацию о расположении программы **child** получает, сканируя массив параметров среды, переданный в третьем параметре функции **main()**.

3) Символ «**&**» порождает дочерний процесс аналогично предыдущему случаю, однако информацию о расположении программы **child** получает, сканируя массив параметров среды, указанный во внешней переменной **extern char **environ**, установленной хост-средой при запуске (см. IEEE Std 1003.1-2017).

При запуске дочернего процесса ему передается сокращенное окружение, включающее набор переменных, указанных в файле, который передается родительскому процессу как параметр командной строки. Минимальный набор переменных должен включать **SHELL**, **HOME**, **HOSTNAME**, **LOGNAME**, **LANG**, **TERM**, **USER**, **LC_COLLATE**, **PATH**. Дочерний процесс открывает этот файл, считывает имена переменных, получает из окружения их значение и выводит в **stdout**.

Дочерний процесс (программа **child**) выводит свое имя, **pid**, **ppid**, открывает файл с набором переменных, считывает их имена, получает из окружения, переданного ему при запуске, их значение способом, указанным при обработке нажатий, выводит в **stdout** и завершается.

Символ «**q**» завершает выполнение родительского процесса.

Программы компилируются с ключами

```
-W -Wall -Wno-unused-parameter -Wno-unused-variable -std=c11 -pedantic
```

Для компиляции, сборки и очистки используется **make**.

№ 3. Взаимодействие и синхронизация процессов

Синхронизация процессов с помощью сигналов и обработка сигналов таймера.

Управление дочерними процессами и упорядочение вывода в **stdout** от них, используя сигналы **SIGUSR1** и **SIGUSR2**.

Действия родительского процесса

По нажатию клавиши «**+**» родительский процесс (P) порождает дочерний процесс (C_k) и сообщает об этом.

По нажатию клавиши «**-**» P удаляет последний порожденный C_k, сообщает об этом и о количестве оставшихся.

При вводе символа «**l**» выводится перечень родительских и дочерних процессов.

При вводе символа «**k**» P удаляет все C_k и сообщает об этом.

При вводе символа «**s**» P запрещает всем C_k выводить статистику (см. ниже).

При вводе символа «**g**» P разрешает всем C_k выводить статистику.

При вводе символов «**s<num>**» P запрещает C_<num> выводить статистику.

При вводе символов «**g<num>**» P разрешает C_<num> выводить статистику.

При вводе символов «**p<num>**» P запрещает всем C_k вывод и запрашивает C_<num> вывести свою статистику. По истечению заданного времени (5 с, например), если не введен символ «g», разрешает всем C_k снова выводить статистику.

По нажатию клавиши «**q**» P удаляет все C_k, сообщает об этом и завершается.

Действия дочернего процесса

Дочерний процесс во внешнем цикле заводит будильник (**nanosleep(2)**) и входит в вечный цикл, в котором заполняет структуру, содержащую пару переменных типа **int**, значениями {0, 0} и {1, 1} в режиме чередования.

При получении сигнала от будильника проверяет содержимое структуры, собирает статистику и повторяет тело внешнего цикла.

Через заданное количество повторений внешнего цикла (например, через 101) дочерний процесс, если ему разрешено, выводит свои PPID, PID и 4 числа — количество разных пар, зарегистрированных в момент получения сигнала от будильника.

Вывод осуществляется посимвольно (**fputc(3)**).

C_k запрашивает доступ к **stdout** у P и осуществляет вывод после подтверждения. По завершению вывода C_k сообщает P об этом.

Следует подобрать интервал времени ожидания и количество повторений внешнего цикла, чтобы статистика была значимой.

Сообщения выводятся в **stdout**.

Сообщения процессов должны содержать идентифицирующие их данные, чтобы можно было фильтровать вывод утилитой **grep**.

№ 4. Задача производители-потребители для процессов

Основной процесс создает очередь сообщений, после чего ожидает и обрабатывает нажатия клавиш, порождая и завершая процессы двух типов —производители и потребители.

Очередь сообщений представляет собой кольцевой буфер, содержащий указатели на сообщения, и пара указателей на голову и хвост. Помимо этого очередь содержит счетчик добавленных сообщений и счетчик извлеченных.

Производители формируют сообщения и, если в очереди есть место, помещают их туда.

Потребители, если в очереди есть сообщения, извлекают их оттуда, обрабатывают и освобождают с ними связанную память.

Для работы используются два семафора для заполнения и извлечения, а также мьютекс или одноместный семафор для монопольного доступа к очереди.

Сообщения имеют следующий формат (размер и смещение в байтах):

Имя	Размер	Смещение	Описание
type	1	0	тип сообщения
hash	2	1	контрольные данные
size	1	3	длина данных в байтах (от 0 до **256**)
data	**((size + 3)/4)*4**	4	данные сообщения

Производители генерируют сообщения, используя системный генератор **rand(3)** для **size** и **data**. В качестве результата для **size** используется остаток от деления на 257.

Если остаток от деления равен нулю, **rand(3)** вызывается повторно. Если остаток от деления равен 256, значение **size** устанавливается равным 0, реальная длина сообщения при этом составляет 256 байт.

При формировании сообщения контрольные данные формируются из всех байт сообщения. Значение поля **hash** при вычислении контрольных данных принимается равным нулю. Для расчета контрольных данных можно использовать любой подходящий алгоритм на выбор студента.

После помещения значения в очередь перед освобождением мьютекса очереди производитель инкрементирует счетчик добавленных сообщений. Затем после поднятия семафора выводит строку на **stdout**, содержащую помимо всего новое значение этого счетчика.

Потребитель, получив доступ к очереди, извлекает сообщение и удаляет его из очереди. Перед освобождением мьютекса очереди инкрементирует счетчик извлеченных сообщений. Затем после поднятия семафора проверяет контрольные данные и выводит строку на **stdout**, содержащую помимо всего новое значение счетчика извлеченных сообщений.

При получении сигнала о завершении процесс должен завершить свой цикл и только после этого завершиться, не входя в новый.

№ 5 Потоки исполнения, взаимодействие и синхронизация

Здесь две задачи. Обе «производители-потребители» для потоков.

5.1) Аналогична лабораторной No 4, но только с потоками, семафорами и мьютексом в рамках одного процесса.

Дополнительно обрабатывается еще две клавиши – увеличение и уменьшение размера очереди.

5.2 Аналогична лабораторной № 5.1, но с использованием условных переменных (см. лекции СПОВМ/ОСиСП).

Изучаемые системные вызовы (префикс **pthread_** опущен): **cond_init()**, **cond_destroy()**, **cond_*wait(), cond_signal()**.

№ 6. Работа с файлами, отображенными в память

Кооперация потоков для высокопроизводительной обработки больших файлов. Изучаемые системные вызовы: **pthread_create()**, **pthread_exit()**, **pthread_join()**, **pthread_yield()**, **pthread_cancel()**, **pthread_barrier_init()**, **pthread_barrier_destroy()**, **pthread_barrier_wait()**, **mmap()**, **munmap()**.

Задание

Написать многопоточную программу **sort_index** для сортировки вторичного индексного файла таблицы базы данных, работающую с файлом в двух режимах: **read()**/**write()** и с использованием отображение файлов в адресное пространство процесса. Программа должна запускаться следующим образом:

```
$ sort_index memsize granul threads filename
```

Параметры командной строки:

```
memsize  := размер рабочего буфера, кратный размеру страницы (getpagesize())
blocks   := порядок разбиения буфера
threads  := количество потоков (от k до N)
k        := количество ядер
N        := максимальное количество потоков (8k??)
filename := имя файла
```

Количество блоков должно быть степенью двойки и превышать количество потоков.

Для целей тестирования написать программу генерации неотсортированного индексного файла.

Алгоритм программы генерации

Генерируемый файл представляет собой вторичный индекс по времени и состоит из заголовка и индексных записей фиксированной длины.

Индексная запись имеет следующую структуру:

```
struct index_s {
    double   time_mark; // временная метка (модифицированная юлианская дата)
    uint64_t recno;     // первичный индекс в таблице БД
} index_record;
```

Заголовок представляет собой следующую структуру

```
struct index_hdr_s {
    uint64_t       recsords; // количество записей
    struct index_s idx[];    // массив записей в количестве records
}
```

Временная метка определяется в модифицированный юлианских днях.

Целая часть лежит в пределах от 15020.0 (1900.01.01-0:0:0.0) до «вчера»[1].

---

1 https://en.wikipedia.org/wiki/Julian_day. Находим в таблице вариантов «Modified JD» и получаем значение даты на сегодня. вычитаем единицу и целую часть используем как максимальное значение целой части генерируемой даты.

Дробная – это часть дня (0.5 – 12:0:0.0). Для генерации целой и дробной частей временной метки используется системный генератор случайных чисел (**`random(3)`**).

Первичный индекс, как вариант, может заполняться последовательно, начиная с 1, но может быть случайным целым > 0 (в программе сортировки не используется).

Размер индекса в записях должен быть кратен 256 и кратно превышать планируемую выделенную память для отображения. Размер индекса и имя файла указывается при запуске программы генерации.

Алгоритм программы сортировки

1) Основной поток запускает **`threads`** потоков, сообщая им адрес буфера, размер блока **`memsize/blocks`**, и их номер от 1 до **`threads - 1`**, используя возможность передачи аргумента для **`start_routine`**. Порожденные потоки останавливаются на барьере, ожидая прихода основного.

2) Основной поток с номером 0 открывает файл, отображает его часть размером **`memsize`** на память и синхронизируется на барьере. Барьер «открывается» и все **`threads`** потоков входят на равных в фазу сортировки.

3) Фаза сортировки

С каждым из блоков связана карта (массив) отсортированных блоков, в которой изначально блоки с 0 по **`threads-1`** отмечены, как занятые.

Поток **n** начинает с того, что выбирает из массива блок со своим номером и его сортирует, используя **qsort(3)**. После того, как поток отсортировал свой первый блок, он на основе конкурентного захвата мьютекса, связанного с картой, получает к ней эксклюзивный доступ, отмечает следующий свободный блок, как занятый, освобождает мьютекс и приступает к его сортировке.

Если свободных блоков нет, синхронизируется на барьере. После прохождения барьера все блоки будут отсортированы.

4) Фаза слияния

Поскольку блоков степень двойки, слияния производятся парами в цикле.

Поток 0 сливает блоки 0 и1, поток 1 – блоки 2 и 3, и так далее.

Для отметки слитых пар и не слитых используется половина карты. Если для потока нет пары слияния, он синхронизируется на барьере.

В результате слияния количество блоков, подлежащих слиянию сокращается в два раза, а размер их в два раза увеличивается.

После очередного прохождения барьера количество блоков, подлежащих слиянию, станет меньше количества потоков. В этом случае распределение блоков между потоками осуществляется на основе конкурентного захвата мьютекса, связанного с картой. Потоки, котором не досталось блока, синхронизируются на барьере.

Когда осталась последняя пара, все потоки с номером не равным нулю синхронизируются на барьере, о поток с номером 0 выполняет слияние последней пары.

После слияния буфер становится отсортирован и подлежит сбросу в файл (**`munmap( )`**).

Если не весь файл обработан, продолжаем с шага 2).

Если весь файл обработан, основной поток отправляет запрос отмены порожденным потокам, выполняет слияние отсортированных частей файла и завершается.

№ 7. Блокировки чтения/записи и условные переменные

Конкурентный доступ к совместно используемому файлу, используя блокировку чтения-записи. Изучаемые системные вызовы: **`fcntl(F_GETLK, F_SETLK, F_SETLKW, F_UNLK)`**.

Программа в режиме конкурентного доступа читает из и пишет в файл, содержащий записи фиксированного формата. Формат записей произвольный. Примерный формат записи:

```
struct record_s {
    char     name[80];    // Ф.И.О. студента
    char     address[80]; // адрес проживания
    uint8_t  semester;    // семестр
}
```

Файл должен содержать не менее 10 записей. Создается и наполняется с помощью любых средств.

Программа должна выполнять следующие операции:

1) **`LST`** – Отображение содержимого файла с последовательной нумерацией записей
2) **`GET(Rec_No)`** – получение записи с порядковым номером **`Rec_No`**;
3) Модификацию полей записи
4) **`PUT`** – сохранение последней прочитанной и модифицированной записи по месту.

Интерфейс с пользователем на «вкус» студента.

Алгоритм конкурентного доступ к записи

```
    REC <-- get(Rec_No)             // читаем запись
Again:
    REC_SAV <-- REC                 // сохраним копию
    /* делаем что-нибудь с записью и желаем ее сохранить */
    if (REC модифицирована) {
        lock(Rec_No)                // блокируем запись для модификации в файле
        REC_NEW <-- get(Rec_No)     // и перечитываем
        if (REC_NEW != REC_SAV) {   // если кто-то изменил запись после
            unlock(Rec_No)          // получения ее нами, освобождаем запись и
            REC <--  REC_NEW        // если требуется,
            goto Again              // повторим все с ее новым содержимым
        }
        put(REC, Rec_No)            // сохраняем новое содержимое
        unlock(Rec_No)              // освобождаем запись
    }
```

Для отладки и тестирования используется не менее двух экземпляров программы.

№ 8. Сокеты. Взаимодействие процессов.

Задача – разработка многопоточного сервера и клиента, работающих по простому протоколу.

Изучаемые системные вызовы: **`socket()`**, **`bind()`**, **`listen()`**, **`conect()`**, **`accept()`** и прочих, связанных с адресацией в домене AF_INET.

Протокол должен содержать следующие запросы:

`ECHO` – эхо-запрос, возвращающий без изменений полученное от клиента;

`QUIT` – запрос на завершение сеанса;

`INFO` – запрос на получения общей информации о сервере;

`CD` – изменить текущий каталог на сервере;

`LIST` – вернуть список файловых объектов из текущего каталога.

Протокол может содержать дополнительные запросы по выбору студента, не выходящие за пределы корневого каталога сервера и не изменяющих файловую систему в его дереве.

Запросы клиенту отправляются на **`stdin`**.

Ответы сервера и ошибки протокола выводятся на **`stdout`**.

Ошибки системы выводятся на **`stderr`**.

Подсказка клиента для ввода запросов – символ **`'>'`**.

Сервер принимает номер порта, на котором будет слушать запросы на соединение, свой корневой каталог и выводит протокол работы в **stdout**.

```
$ myserver port_no dir
Готов.
2024.03.17-14:25:17.234 conn_req 192.168.11.234 accepted port:14367
...
```

Формат протокола произвольный, каждое событие занимает одну строку, первое поле – дата и время в формате YYYY.MM.DD-hh:mm:ss.sss).

Клиент помимо интерактивных запросов принимает запросы из файла. Файл с запросами указывается с использованием префикса `'@'`:

```
$ myclient server port
Вас приветсвует учебный сервер 'myserver'
> @file
> ECHO какой-то_текст
какой-то_текст
> LIST
dir1
dir2
file
> CD dir1
dir1> QUIT
BYE
$
```

ECHO – эхо-запрос, возвращающий без изменений полученное от клиента.

```
> ECHO "произвольный текст"
произвольный текст
>
```

QUIT – запрос на завершение сеанса

```
> QUIT
BYE
$
```

INFO – запрос на получения общей информации о сервере.

Сервер отправляет текстовый файл с соответствующей информацией.

```
> INFO
Вас приветствует учебный сервер 'myserver'
>
```

Этот же файл сервер отправляет клиенту при установлении сеанса.

LIST – вернуть список файловых объектов из текущего каталога.

Текущий каталог – каталог в дереве каталогов сервера. Корневой каталог сервера устанавливается из командной строки при старте сервера.

```
> LIST
dir1/
dir2/
file1
file2 --> dir2/file2
file3 -->> dir1/file
>
```

Каталоги выводятся с суффиксом `'/'` после имени, файлы – как есть, симлинки на регулярные файлы разрешаются через **`'-->'`**, симлинки на симлинки разрешаются через **`'-->>'`**. Корневой каталог сервера пр выводе указывается префиксом `'/'` перед именем.

CD – изменить текущий каталог на сервере

Выход за пределы дерева корневого каталога сервера запрещается, команда безмолвно игнорируется

```
> CD dir2
dir2> LIST
file2
dir2> CD ../dir1
dir1> LIST
file --> /file1
dir1> CD ..
> CD ..
>
```

Соединения функционируют независимо, т.е. текущий каталог у каждого соединения свой.

Примечания:

Раскрашивать вывод не нужно.

Для разработки и отладки лабораторной следует использовать редактор или IDE, поддерживающие несколько запущенных экземпляров, каждый со своей конфигурацией, и поддерживающие отладку прямо в окне с кодом, например, slickedit (лучший выбор).

Курсовая работа, ее характеристика

Курсовой проект по данному предмету — это самостоятельная программная разработка студента по заданной теме в области системного программирования.

Целью – приобретение навыков в реализации цикла разработки ПС:

- техническое задание на проект;
- анализ предметной области;
- разработка алгоритмов;
- реализация программного средства;
- отладка и тестирование;
- оформление пояснительной записки и чертежа форматом А1.

Все в соответствии с требованиями действующих стандартов.

Возможно выполнение как индивидуальных, так и коллективных курсовых работ (2-3 человека). Курсовая работа выполняется с использованием языков программирования C и C++ (gcc, g++) и языка ассемблера (nasm) для платформы Linux.

В состав курсового проекта входят:

- пояснительная записка;
- графическая часть;
- работающее программное средство.

Пояснительная записка должна отражать основные этапы разработки программного средства.

1) Многопоточная программа для фонового контроля изменений и целостности группы файлов;

2) Многопоточная переносимая программа обмена файлами (c, c++);

3) Утилита сбора информации о системе (из /proc, ncurses);

4) Простая файловая система (SFS) в пространстве пользователя;

5) Утилита форматирования и проверки файловой системы SFS;

6) Низкоуровневый редактор блочного устройства уровня секторов (ncurses);

7) Анализатор и редактор файловой системы (ncurses);

8) Утилита обнаружения и тестирования функций USB-устройства;

9) FTP-сервер с возможностью получения/отправки архивированных каталогов;

10) FTP-клиент с возможностью получения/отправки архивированных каталогов;

11) Разработка и реализация протокола аутентификации клиента на сервере с использованием функционала ssl;

12) Разработка и реализация низколатентного протокола сбора неоднородных телеметрических данных от нескольких источников в TCP/IP сети (клиент);

13) Разработка и реализация имитационной модели источника неоднородных телеметрических данных (сервер);

14) Диспетчер параллельных процессов обработки набора файлов с помощью внешних программ;

15) «Корзина» для программ, использующих системный вызов **`unlink()`**;

16) Программа-демонстратор технологии использования стандартных совместно используемых библиотек **`.so`**, **`.dll`** из управляемого кода (mono, C#).

17) Утилита контроля появления дубликатов в файловой системе с заменой их на жесткие ссылки и протоколирования фактов замены.

18) Разработка субаллокатора памяти в пространстве процесса для обновляемого индекса на основе B-дерева (c, c++).

19) Разработка двухпанельного файлового менеджера (ncurses) с вкладками.

20) Разработка эмулятора арифметических инструкций с фиксированной запятой.

21) Разработка симулятора контроллера без инструкции деления (nasm).

22) Разработка ассемблера для симулятора контроллера и библиотеки поддержки целочисленных операций.

23) Разработка кроссассемблера для симулятора контроллера.

24) Разработка симулятора процессора мини-ЭВМ PDP-11 (nasm, c).

25) Разработка ассемблера для симулятора процессора PDP-11 (c, c++).

26) Разработка симулятора DSP slice Virtex 5 (nasm, c).

27) Разработка симулятора DSP slice Virtex 7 (nasm, c).

28) Программа-сторож неправильной раскладки русский/английский с заменой введенного фрагмента и переключением раскладки.

29) ncurses-оболочка для утилиты find с хранением и редактированием истории запросов.

30) Утилита коррекции дат в архивах по максимальной из выбранных типов файлов.

31) Программа-аналог fdupes с ограничением поиска и анализа по сигнатурам и mime-типам файлов.

32) Разработка симулятора ЭВМ на базе стекового процессора.

33) Разработка простого ассемблера и загрузчика для стекового процессора.