Операционные системы и системное программирование

Лабораторные работы

Преподаватель: Поденок Леонид Петрович

к. 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, 2023

Оглавление

Общие замечания

Требования к проекту лабораторной работы
Проект должен компилироваться и собираться gcc без предупреждений.
Обязательные опции gcc:

```
-std=c11 -pedantic -W -Wall -Wextra
```

Проект лабораторной работы должен располагаться в отдельном каталоге и содержать:
- исходные файлы на языке С с комментариями на русском языке в кодировке utf8;
- makefile – файл управления сборкой проекта;
- краткое описание проекта в текстовом формате на русском языке в кодировке utf8;
- скрипты, входные или иные данные, необходимые для выполнения программ проекта;
- отчет о лабораторной работе (.pdf).

Файл управления сборкой проекта должен содержать следующие цели;
clean – очистка каталога с проектом от результатов сборки;
all – сборка проекта целиком.

Каталог проекта архивируется снаружи архиватором tar. Допускается использовать сжатие в форматы, которые поддерживаются архиватором, например gzip.

При разворачивании архива в текущем каталоге должна создаваться исходная файловая структура проекта лабораторной.

Наличие иных файлов, в том числе размещаемых операционной системой для целей индексации и прочих, не допускается.

Для справки примерный состав каталога с проектом лабораторной и его архивом

```
$ find
.
./lab03.tar.gz
./lab03
./lab03/lab03.c
./lab03/consumer.c
./lab03/lab03.txt
./lab03/producer.c
./lab03/lab03.pdf
./lab03/makefile
$ tar -zcf lab03.tar.gz lab03
$ tar tf lab03.tar.gz
lab03/
lab03/lab03.c
lab03/consumer.c
lab03/lab03.txt
lab03/producer.c
lab03/lab03.pdf
lab03/makefile
```

Лабораторные

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

Оболочка `bash`, файловый менеджер `mc`, стандартное информационное обеспечение (`info`, `man`).

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`, lab03, `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)`.

Лабораторная работа No 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.

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.

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

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`.

Лабораторная работа No 3. Взаимодействие и синхронизация процессов

Синхронизация процессов с помощью сигналов и обработка сигналов таймера.

Задание

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`.

Лабораторная работа No 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 классическую структуру — 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, перемещают их туда.

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

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.

Следует предусмотреть задержки, чтобы вывод можно было успеть прочитать в процессе работы программы.

Следует предусмотреть защиту от тупиковых ситуаций из-за отсутствия производителей или потребителей.

Лабораторная работа No 5. Потоки исполнения, взаимодействие и синхронизация

Задача производители-потребители для потоков. Аналогична лабораторной No 4, но только с потоками в рамках одного процесса.

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

Лабораторная работа No 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]. 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 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.

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.

Лабораторная работа No 7. Блокировки чтения/записи и условные переменные

Здесь две программы.

1) Задача «производители-потребители». Аналогична лабораторной № 4, но для потоков с использованием условных переменных (см. лекции СПОВМ).

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

2) 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)
Again:
    REC_SAV <-- REC               // сохраним копию
    if (REC модифицирована) {
        lock(Rec_No)              // блокируем запись для модификации в файле
        REC_NEW <-- get(Rec_No)   // и перечитываем
        if (REC_NEW != REC_SAV) { // кто-то изменил запись после получения ее нами
            unlock(Rec_No)        // освобождаем запись и
            REC <--  REC_NEW      // повторим все с ее новым содержимым
            goto Again
        }
        put(REC, Rec_No)          // сохраняем новое содержимое
        unlock(Rec_No)            // освобождаем запись
    }
```

Для отладки и тестирования используется не менее двух экземпляров программы.

Лабораторная работа No 8. Сокеты. Взаимодействие процессов.

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

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

Протокол должен содержать не менее запростов, среди которых должны быть:

ECHO – эхо-запрос, возвращающий