

Системное программное обеспечение вычислительных машин (СПОВМ)

Лабораторные работы

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, 2021

Оглавление

Общие замечания.....	3
Лабораторные.....	5
Лабораторная работа No 1. Знакомство с Linux (Unix) и средой программирования.....	5
Лабораторная работа No 2. POSIX-совместимая файловая система.....	6
Лабораторная работа No 3. Понятие процессов.....	7
Лабораторная работа No 4. Взаимодействие и синхронизация процессов.....	9
Лабораторная работа No 5. Потоки исполнение (threads).....	11
Лабораторная работа No 6. Взаимодействие и синхронизация потоков.....	12
Лабораторная работа No 7. Задача производителя-потребителя для процессов.....	13
Лабораторная работа No 8. Задача производителя-потребителя для потоков.....	14
Лабораторная работа No 9. Работа с файлами, отображенными в память.....	15
Лабораторная работа No 10. Блокировки чтения/записи и условные переменные.....	16
Лабораторная работа No 11. Сокеты. Взаимодействие процессов.....	17

Общие замечания

Требования к проекту лабораторной работы

Проект должен компилироваться и собираться gcc без предупреждений.

Опции gcc:

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

Проект лабораторной работы должен располагаться в отдельном каталоге и содержать:

- исходные файлы на языке C с комментариями на русском языке (utf8);
- makefile – файл управления сборкой проекта;
- краткое описание проекта в текстовом формате (utf8);
- скрипты, входные или иные данные, необходимые для выполнения программ проекта;
- отчет о лабораторной работе (.pdf).

Каталог проекта архивируется снаружи в `.tar.gz`, при разворачивании которого в текущем каталоге должна создаваться исходная файловая структура.

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

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 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

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`, `pwd`, `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`,

Задание

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

Лабораторная работа No 2. POSIX-совместимая файловая система

Структура ФС, содержимое inode, команды оболочки

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

Задание

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`

Если опции `ldf` опущены, выводятся каталоги, файлы, ссылки, как у `find` без параметров.

Лабораторная работа No 3. Понятие процессов.

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

Создаются 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 «+» порождает дочерний процесс, используя `fork()` и `execve()`, запускает очередной экземпляр программы `child`, используя информацию о каталоге из окружения, которую получает, 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 его расположении получает, сканируя массив параметров среды, переданный в третьем параметре функции `main()`.

Символ «&» порождает дочерний процесс аналогично предыдущему случаю, однако информацию о его расположении получает, сканируя массив параметров среды, указанный в переданный 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`, `HOSTNAME`, `LOGNAME`, `HOME`, `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 «&» 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 4. Взаимодействие и синхронизация процессов

Синхронизация процессов с помощью сигналов и обработка сигналов таймера.

Задача — 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 «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 в режиме чередования заполняет структуру, содержащую пару переменных типа int, значениями {0, 0} и {1, 1} (см. раздел «Проблемы неатомарного доступа» темы «Сигналы»).

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.

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.

Лабораторная работа No 5. Потоки исполнение (threads)

Лабораторная работа No 6. Взаимодействие и синхронизация потоков

Аналогична лабораторной No 4, но только с потоками в рамках одного процесса.

Лабораторная работа No 7. Задача производители-потребители для процессов

Лабораторная работа No 8. Задача производители-потребители для потоков

Лабораторная работа No 9. Работа с файлами, отображенными в память

Лабораторная работа No 10. Блокировки чтения/записи и условные переменные

Лабораторная работа No 11. Сокеты. Взаимодействие процессов.