Министерство образования Республики Беларусь Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет: компьютерных систем и сетей

Кафедра: ЭВМ

Дисциплина: Операционные системы и системное программирование

ОТЧЁТ
к лабораторной работе
№3 на тему
Взаимодействие и синхронизация процессов.

Выполнил студент гр.230501 Кочеров Р.С.

Проверил старший преподаватель кафедры ЭВМ Поденок Л.П.

Минск 2024

1 УСЛОВИЕ ЛАБОРАТОРНОЙ РАБОТЫ.

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. Поскольку заполнение не атомарно, в момент срабатывании будильника в структуре может оказаться любая возможная комбинация из 0 и 1.

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 в одну строку.

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.

2 ОПИСАНИЕ АЛГОРИТМОВ И РЕШЕНИЙ.

Программа предназначена для запуска родительского процесса parent, процесс дает возможность работать с дочерними процессами через различные операции , которые обрабатываются при нажатии.

Основные библиотечные функции которые использовались в лабораторной — `execve()`, `kill()`, `fork()`, `getpid()`, `getppid()` , `signal()`.

2.1 Программа parent

Данную программу мы запускаем, для того чтобы запустить дочерний процесс. :

Функция `main()` в предоставленном коде служит точкой входа программы и реализует основную логику. Вот разбор алгоритма, выполняемого в функции main():

1) Вызывается функция `onSignal()`, чтобы установить обработчик сигнала для сигнала SIGALRM, который будет использоваться для обработки событий сигналов.

2) Программа входит в бесконечный цикл, используя while (true).

3) Внутри цикла вызывается функция `readCommand()`, чтобы считать команду из ввода пользователя. Команда сохраняется в массив `commandBuffer`.

4) Вызывается функция `handleCommand()`, передавая `commandBuffer` в качестве аргумента. Эта функция анализирует команду пользователя и выполняет соответствующее действие.

5) Если функция `handleCommand()` возвращает false, основной цикл завершается.

6) Наконец, программа вызывает `killAllProcesses()` и возвращает соответствующий код выхода (0, если все процессы были успешно завершены, или ненулевое значение в противном случае).

Функция main() служит центральным потоком управления программы, управляя выполнением команд пользователя и жизненным циклом дочерних процессов.

Другие функции в коде обеспечивают следующую функциональность:

1) `executeProcess()`: Создает новый дочерний процесс и выполняет указанную программу с заданными аргументами и окружением.

2) `startChildProcess()`: Запускает новый дочерний процесс, вызывая executeProcess() с программой "build/child".

3) `createNewProcess()`, `killLastProcess()`, `killAllProcesses()`, `listProcesses()`: Управляют созданием, завершением и списком дочерних процессов.

4) `SilenceProcess()`, `unsilenceProcess()`, `silence-AllProcesses()`, `unsilenceAllProcesses()`: Управляют приглушением и включением звука дочерних процессов.

5) `prioritizeProcess()`: Приглушает все дочерние процессы и включает звук указанного процесса, устанавливая сигнал тревоги для включения звука всех процессов через 5 секунд.

6) `quit()`: Обрабатывает команду для выхода из программы.

7) `unknownCommand()`: Обрабатывает неизвестные команды пользователя.

8) `onAlarmHandler()`: Обрабатывает сигнал тревоги, включая звук всех дочерних процессов.

9) `handleCommand()`: Анализирует и выполняет команды пользователя.

10) `IsFastCommand()`, `getInputCharacter()`, `append CharacterToString()`, `readCommand()`: Вспомогательные функции для чтения и обработки ввода пользователя.

В коде представлены следующие команды:

Команда q: завершает все дочерние процессы, убивая их сигналом SIGKILL, и завершает программу.

Команда +: создает новый дочерний процесс с помощью функции fork и вызывает execve для запуска программы ./child.

Команда -: убивает последний созданный дочерний процесс с помощью сигнала SIGKILL.

Команда l: выводит информацию о родительском процессе и созданных дочерних процессах.

Команда k: завершает все дочерние процессы, убивая их сигналом SIGKILL.

Команда s: отправляет сигнал SIGUSR1 указанному дочернему процессу или всем дочерним процессам.

Команда g: отправляет сигнал SIGUSR2 указанному дочернему процессу или всем дочерним процессам.

Команда p: отправляет сигнал SIGUSR1 указанному дочернему процессу и ожидает ввода с клавиатуры в течение 5 секунд. Если вводится символ "g", отправляется сигнал SIGUSR2 указанному дочернему процессу.

Для выполнения команды p, используется функция select, которая позволяет ожидать ввода на стандартный ввод (stdin) в течение определенного времени. Если происходит ввод в течение 5 секунд, программа проверяет введенный символ. Если это символ "g", отправляется сигнал SIGUSR2 указанному дочернему процессу.

После выполнения каждой команды программа возвращается в начало цикла while и ожидает новую команду от пользователя.

2.2 Программа child

1) Вызывается функция onSignal(), чтобы установить обработчики сигналов для сигналов SIGUSR1 и SIGUSR2, которые будут вызывать функции onUsr1() и onUsr2() соответственно, когда эти сигналы будут получены.

2) Вызывается функция startIntervalTimer(), которая создает таймер, который будет вызывать функцию onTimer() каждые 500 микросекунд (0,5 миллисекунд).

3) Программа входит в бесконечный цикл с использованием конструкции while (true).

4) Внутри цикла вызывается функция reset() для инициализации структур coordinates и counter_values их значениями по умолчанию (все поля установлены в 0).

5) Затем выполняется цикл 100 000 000 раз, где поля coordinates.x и coordinates.y сначала устанавливаются в 0, а затем в 1.

6) После цикла вызывается функция print() для вывода

текущих значений структуры counter_values, если только флаг is_silent не установлен в true.

7)Программа затем переходит к следующей итерации внешнего цикла while (true).

Другие функции в коде имеют следующие обязанности:

1) `reset()`: Инициализирует структуры coordinates и counter_values их значениями по умолчанию (все поля установлены в 0).

2) `collect()`: Анализирует текущее состояние структуры coordinates и обновляет соответствующие поля в структуре counter_values.

3) `print()`: Выводит текущие значения структуры counter_values, включая идентификатор процесса (PID) и идентификатор родительского процесса (PPID).

4) `onTimer()`: Эта функция вызывается каждые 500 микросекунд таймером, установленным в функции main(), и она вызывает функцию collect() для обновления структуры counter_values.

5) `onUsr1()`: Эта функция вызывается, когда программа получает сигнал SIGUSR1, и она устанавливает флаг is_silent в true, заставляя функцию print() не выводить никакой информации.

6) `onUsr2()`: Эта функция вызывается, когда программа получает сигнал SIGUSR2, и она устанавливает флаг is_silent в false, заставляя функцию print() снова выводить информацию.

3 ФУНКЦИОНАЛЬНАЯ СТРУКТУРА ПРОЕКТА.

Проект собирается с помощью makefile. Пример запуска:

```
rlinux@fedora:~$/home/rlinux/Kocherov/ОСиСП/lab03/build/parent
```

Для запуска проекта нам требуется в терминале заупустить программу parent. Где программа сразу переходит в цикл обработки символов

В проекте имеется каталог для сборки build. Каталог git для

системы контроля версий моего проекта. Директория src с исходным кодом. И makefile для компиляции и сборки проекта.

4 ПОРЯДОК СБОРКИ И ИСПОЛЬЗОВАНИЯ.

Для компиляции и сборки проекта используется makefile.

Порядок сборки:

1) Определение переменных:
   rwildcard - рекурсивно находит все файлы по заданному шаблону
   eq - определяет, равны ли два значения
   TARGET - целевой исполняемый файл
   BUILD_DIR - каталог для сборки
   DEBUG_SUFFIX - суффикс для отладочной сборки
   CFLAGS - флаги компиляции
   CC - компилятор с флагами
   VALGRIND и GDB - инструменты для отладки
   GDB_COMMANDS - файл с командами для GDB

2) Определение компиляторов и флагов компиляции:
   CFLAGS содержит флаги компиляции, в том числе для отладочной
сборки
   CC определяет компилятор с флагами

3) Определение зависимостей:
   SOURCES - список исходных файлов
   OBJECTS - список объектных файлов
   MAINS - список главных файлов
   EXECUTABLES - список исполняемых файлов

4) Определение целей:
   all - цель для сборки всех исполняемых файлов
   run - цель для запуска целевого исполняемого файла
   vrun - цель для запуска целевого исполняемого файла под Valgrind
   gdb - цель для запуска целевого исполняемого файла под GDB
   app - цель для сборки всех исполняемых файлов
   clean - цель для очистки директории сборки

Порядок использования.

1) Компиляция:

Для релизной сборки: make

2) Очистка:

make clean - удаляет все объектные файлы и исполняемый файл.

3) Запуск:

После успешной компиляции запустите исполняемый файл: ./build/parent.

5 МЕТОД ТЕСТИРОВАНИЯ И РЕЗУЛЬТАТ ТЕСТИРОВАНИЯ.

```
rlinux@fedora:~/Kocherov/ОСиСП/lab03$ build/parent
Created new process: 11842
ppid:11828 pid:11842 down_down: 70 down_up: 61 up_down:
74 up_up: 560
ppid:11828 pid:11842 down_down: 70 down_up: 73 up_down:
7
78 up_up: 538
ppid:11828 pid:11842 down_down: 41 down_up: 62 up_down:
73 up_up: 599
ppid:11828 pid:11842 down_down: 66 down_up: 76 up_down:
74 up_up: 551
ppid:11828 pid:11842 down_down: 71 down_up: 67 up_down:
54 up_up: 582
ppid:11828 pid:11842 down_down: 62 down_up: 82 up_down:
65 up_up: 552
ppid:11828 pid:11842 down_down: 61 down_up: 73 up_down:
79 up_up: 546
Created new process: 17762
ppid:11828 pid:11842 down_down: 63 down_up: 63 up_down:
60 up_up: 586
ppid:11828 pid:17762 down_down: 62 down_up: 68 up_down:
70 up_up: 658
ppid:11828 pid:11842 down_down: 68 down_up: 64 up_down:
68 up_up: 568
ppid:11828 pid:17762 down_down: 67 down_up: 84 up_down:
89 up_up: 601
ppid:11828 pid:11842 down_down: 47 down_up: 80 up_down:
59 up_up: 579
ppid:11828 pid:11842 down_down: 61 down_up: 71 up_down:
72 up_up: 592
ppid:11828 pid:17762 down_down: 52 down_up: 67 up_down:
61 up_up: 690
lppid:11828 pid:11842 down_down: 52 down_up: 62 up_down:
48 up_up: 614
ppid:11828 pid:17762 down_down: 65 down_up: 74 up_down:
```

```
73 up_up: 633
ppid:11828 pid:11842 down_down: 34 down_up: 39 up_down:
34 up_up: 717
ppid:11828 pid:17762 down_down: 36 down_up: 40 up_down:
28 up_up: 788
ppid:11828 pid:11842 down_down: 46 down_up: 66 up_down:
65 up_up: 615
ppid:11828 pid:17762 down_down: 64 down_up: 71 up_down:
82 up_up: 623
ppid:11828 pid:11842 down_down:
 8 down_up:
 4 up_down:
6 up_up: 778
ppid:11828 pid:17762 down_down: 56 down_up: 77 up_down:
79 up_up: 642
ppid:11828 pid:11842 down_down: 26 down_up: 30 up_down:
20 up_up: 733
ppid:11828 pid:17762 down_down: 34 down_up: 43 up_down:
38 up_up: 794
l
8
Unknown command: ll
ppid:11828 pid:11842 down_down: 47 down_up: 58 up_down:
46 up_up: 666
ppid:11828 pid:17762 down_down: 53 down_up: 80 up_down:
80 up_up: 622
ppid:11828 pid:11842 down_down: 50 down_up: 64 up_down:
70 up_up: 626
ppid:11828 pid:17762 down_down: 56 down_up: 52 up_down:
63 up_up: 698
ppid:11828 pid:11842 down_down: 40 down_up: 49 up_down:
40 up_up: 649
ppid:11828 pid:17762 down_down: 43 down_up: 63 up_down:
76 up_up: 714
ppid:11828 pid:11842 down_down: 56 down_up: 61 up_down:
68 up_up: 592
ppid:11828 pid:17762 down_down: 61 down_up: 79 up_down:
79 up_up: 634
ppid:11828 pid:11842 down_down: 41 down_up: 42 up_down:
61 up_up: 634
ppid:11828 pid:17762 down_down: 66 down_up: 87 up_down:
61 up_up: 557
ppid:11828 pid:11842 down_down: 20 down_up: 21 up_down:
25 up_up: 695
ppid:11828 pid:17762 down_down: 65 down_up: 60 up_down:
62 up_up: 497
ppid:11828 pid:11842 down_down: 34 down_up: 30 up_down:
```

```
28 up_up: 562
ppid:11828 pid:11842 down_down: 41 down_up: 42 up_down:
56 up_up: 404
ppid:11828 pid:17762 down_down: 28 down_up: 38 up_down:
32 up_up: 546
ppid:11828 pid:11842 down_down: 43 down_up: 45 up_down:
q
rlinux@fedora:~/Kocherov/ОСиСП/lab03$
```