Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

Дисциплина: Операционные системы и системное программирование

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ

по теме

ПОНЯТИЕ ПРОЦЕССОВ

БГУИР КР 1-40 02 01 001 ЛР

Выполнил: студент группы 250541,
Бобрик В. Ю.

Проверил: ст.преподаватель каф. ЭВМ,
Поденок Л. П.

Минск 2025

СОДЕРЖАНИЕ

1 Условия лабораторной работы

Изучение системных вызовов fork(), execve(), getpid(), getppid(), среды исполнения, функции getenv(), связи системного вызова execve() с функцией main() языка C.

Задание

Разработать две программы – parent (родительский процесс) и child (дочерний процесс).

Родительский процесс создает сокращенную среду (окружение) для дочернего. Для этого пользователем создается файл env, содержащий небольшой набор имен переменных окружения.

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

Родительский процесс (программа parent) после запуска получает переменные своего окружения и их значения, сортирует в LC_COLLATE=C и выводит в stdout. Читает файл env и формирует среду для дочернего процесса в том виде, в котором она указывается в системном вызове execve(), используя значения для переменных из собственной среды. После этого входит в цикл обработки нажатий клавиатуры.

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 «+»

Используя fork(2) и execve(2) порождает дочерний процесс и запускает в нем очередной экземпляр программы child. Информацию о каталоге, где размещается child, parent получает из своего окружения, используя функцию getenv(). Имя программы child (argv[0]) устанавливается как

child_XX, где XX порядковый номер от 00 до 99 (номер инкрементируется родителем).

Дочерний процесс выводит свое имя, pid и ppid в stdout. Вторым параметром программы child является путь к файлу env, который читается для получения значений параметров среды. Дочерний процесс открывает этот файл, считывает имена переменных, получает из окружения их значение, используя getenv(), и выводит в stdout.

Символ «*»

Дочерний процесс порождается аналогично предыдущему случаю, однако информацию о своем окружении программа child получает, сканируя массив параметров среды, переданный в третьем параметре функции main() и выводит в stdout. Путь к файлу env передавать в параметрах не требуется.

Символ «&»

Дочерний процесс порождается аналогично предыдущему случаю, однако информацию о своем окружении программа child получает, сканируя массив параметров среды, указанный во внешней переменной extern char **environ, установленной хост-средой при запуске (см. IEEE Std 1003.1-2017) и выводит в stdout.

Минимальный набор переменных в файле env должен включать SHELL, HOME, HOSTNAME, LOGNAME, LANG, TERM, USER, LC_COLLATE, PATH.

Символ «q» завершает выполнение родительского процесса.

Требования к сборке

Программы компилируются с ключами

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

Допускается использование ключей -Wno-unused-parameter -Wno-unused-variable.

Для компиляции, сборки и очистки используется make.

2 Описание алгоритмов и решений

Алгоритм родительского процесса состоит из следущих шагов:

Шаг 1. При запуске родительский процесс выводит свой PID и приглашает пользователя ввести символ.

Шаг 2. Пользователь вводит один из следующих символов:

«+» – запуск дочернего процесса в режиме plus.

«*» – запуск дочернего процесса в режиме star.

«&» – запуск дочернего процесса в режиме ampersand.

«q» – завершение работы родительского процесса.

Шаг 3. Функции из модуля spawn_children реализуют следующие алгоритмы:

Шаг 3.1. Режим plus.

Шаг 3.1.1. Вызывается fork() для создания нового процесса.

Шаг 3.1.2. Формируется имя дочернего процесса по схеме child_XX, где XX – порядковый номер.

Шаг 3.1.3. Получается путь к исполняемому файлу child через переменную CHILD_PATH (с помощью getenv()).

Шаг 3.1.4. Вторым аргументом в execve() передаётся путь к файлу env. Окружение передаётся полностью (используется родительское environ).

Шаг 3.2. Режим star.

Шаг 3.2.1. Вызывается fork() для создания нового процесса.

Шаг 3.2.2. Формируется имя дочернего процесса.

Шаг 3.2.3. Создаётся сокращённое окружение с фиксированным набором переменных с добавлением параметра CHILD_MODE=star для идентификации режима.

Шаг 3.2.4. Вызывается execve() с подготовленным окружением и без передачи второго аргумента.

Шаг 3.3. Режим ampersand.

Шаг 3.3.1. Вызывается fork() для создания нового процесса.

Шаг 3.3.2. Формируется имя дочернего процесса.

Шаг 3.3.3. Вызывается execve() с передачей стандартного родительского окружения (глобальная переменная environ). Специальная переменная CHILD_MODE не передается, что позволяет дочернему процессу определить, что нужно использовать режим “&”.

Шаг 3.4. Режим q. Завершение работы.

Алгоритм дочернего процесса состоит в следующем:

Шаг 1. Вывод базовой информации: имя программы (argv[0]), PID и PPID.

Шаг 2. Определение режима работы.

Шаг 3.1. Если передан второй аргумент (путь к файлу env).

Шаг 3.1.1. Открывается файл env.

Шаг 3.1.2. Считываются имена переменных.

Шаг 3.1.3. Для каждой переменной с помощью getenv( ) выводится её значение.

Шаг 3.2. Если второй аргумент отсутствует, но в окружении обнаружена переменная CHILD_MODE.

Шаг 3.2.1. Получение массива envp.

Шаг 3.2.2. Вывод переменных окружения.

Шаг 3.3. Если ни второй аргумент не передан, ни присутствует переменная CHILD_MODE, выводятся переменные из глобальной переменной environ.

Система разделена на отдельные файлы: родительский процесс отделён от логики создания дочерних процессов, а дочерний процесс имеет самостоятельную логику выбора режима. Это упрощает поддержку и тестирование.

Использование разных вариантов окружения обеспечивает гибкость и позволяет дочернему процессу определить, какой режим работы требуется.

Вместо сложных проверок, выбор режима определяется по наличию аргументов и специфической переменной `CHILD_MODE`, что делает логику понятной и устойчивой.

3 Функциональная структура проекта

Проект состоит из следующих основных компонентов:

Программа `parent` – родительский процесс, который занимается инициализацией и выводом собственной информации (`PID` процесса, приглашения для ввода команд), обработкой ввода пользователя, где в зависимости от нажатой клавиши выбирается один из трёх режимов для порождения дочернего процесса, вызовом специализированных функций для создания дочерних процессов (через модуль `spawn_children`).

Программа `child` – дочерний процесс, логика которого определяется на основе переданных аргументов (наличие второго параметра указывает на режим «+»), содержимого переменных окружения (наличием переменной `CHILD_MODE` определяется режим работы «*») или их отсутствия (определяется режим «&»). В каждом случае дочерний процесс выводит свою базовую информацию и список переменных окружения, используя соответствующую стратегию (чтение файла, анализ `envp` или использование глобальной переменной `environ`).

`handle_plus_mode`() – обработчик режима «+».

`handle_star_mode`() – обработчик режима «*».

`handle_amp_mode`() – обработчик режима «&».

Модуль `spawn_children` объединяет функции для порождения дочернего процесса для каждого режима:

`spawn_child_plus`(): Создает дочерний процесс для режима «+».

spawn_child_star(): Создает дочерний процесс для режима «*», формируя специальное сокращенное окружение с переменной CHILD_MODE=star.

spawn_child_amp(): Создает дочерний процесс для режима «&», передавая стандартное окружение родительского процесса.

4 Порядок сборки и тестирования

Для сборки используется Makefile, содержащий следующие ключевые элементы:

Исходные файлы находятся в каталоге ./src.

Заголовочные файлы расположены в каталоге ./include.

В зависимости от типа сборки создаётся каталог ./out/debug для debug-версии или ./out/release для release-версии.

Переменная BUILD задаётся как debug по умолчанию. При запуске make release устанавливается оптимизированная сборка.

Флаги компиляции задаются через переменные COMMON_CFLAGS (определения, предупреждения, стандарт C11) и дополняются флагами отладки (-g -O0) или оптимизации (-O2).

Для сборки в debug-режиме необходимо выполнить в терминале:

```
make
```

Это соберёт программы в ./out/debug/.

Для сборки в release-режиме необходимо выполнить в терминале:

```
make release
```

Это соберёт программы в ./out/release/.

Для очистки сборки необходимо выполнить в терминале:

```
make clean
```

Необходимо настроить окружение для тестирования программы.

Так как программа parent должна знать, где находится исполняемый файл child, необходимо установить переменную окружения CHILD_PATH на путь к каталогу со сборками. Для debug-сборки выполнить:

```
export CHILD_PATH=$(pwd)/out/debug
```

Для release-сборки выполнить:

```
export CHILD_PATH=$(pwd)/out/release
```

В корневом каталоге необходимо создать файл env и заполнить его списком имен переменных.

```
nano env
SHELL
HOME
HOSTNAME
LOGNAME
LANG
TERM
USER
LC_COLLATE
PATH
```

Тестирование осуществлять в ручном режиме. Для запуска программы выполнить:

```
./out/release/parent
```

5 Методика и результаты тестирования

В терминале необходимо запустить родительский процесс:

```
./out/release/parent
```

Появляется следующий вывод:

```
[silvarious@fedora lab02]$ out/release/parent
PID: 1625
Введите '+', '*', '&' для порождения процесса, 'q' для выхода:
```

Проводится пошаговое тестирование каждого из режимов.

Ввести в терминал «+». Ожидается, что родительский процесс создаст дочерний процесс child_00. Дочерний процесс выводит своё имя, PID и PPID. Затем он открывает файл env, считывает имена переменных и выводит каждое значение. Значения должны быть указаны согласно экспортированным ранее.

```
+
Запущен дочерний процесс 0 с PID 1627 (режим '+')
Имя программы: child_00
PID: 1627
PPID: 1625
[Режим '+']: Чтение переменных из файла env и использование getenv()
SHELL=/bin/bash
HOME=/home/test
HOSTNAME=testhost
LOGNAME=testuser
LANG=en_US.UTF-8
TERM=xterm-256color
USER=testuser
LC_COLLATE=C
PATH=/usr/bin:/bin
Дочерний процесс завершен.
```

Ввести в терминал «*». Ожидается, что родитель создаст новый процесс child_01 с передачей сокращенного окружения, в котором дополнительно присутствует переменная CHILD_MODE=star. Дочерний процесс должен определить, что переменная CHILD_MODE присутствует, вывести список переменных, получаемых из переданного массива envp. Следует обратить внимание на наличие переменной CHILD_MODE.

```
*
Запущен дочерний процесс 1 с PID 1644 (режим '*')
Имя программы: child_01
PID: 1644
```

```
PPID: 1625
[Режим '*']: Использование параметра envp
SHELL=/bin/bash
HOME=/home/silvarious
HOSTNAME=fedora
LOGNAME=silvarious
LANG=en_US.UTF-8
TERM=xterm-256color
USER=silvarious
LC_COLLATE=C
PATH=/usr/bin:/bin:/usr/sbin:/sbin
CHILD_MODE=star
Дочерний процесс завершен.
```

Ввести в терминал «&». Ожидается, что родительский процесс создаст дочерний child_02 с использованием стандартного окружения (глобальная переменная environ). Дочерний процесс должен определить режим «&», вывести переменные через глобальную переменную environ. Вывод включает системные и пользовательские переменные, а переменная CHILD_MODE отсутствует.

```
&
Запущен дочерний процесс 2 с PID 1648 (режим '&')
Имя программы: child_02
PID: 1648
PPID: 1625
[Режим '&']: Использование глобальной переменной environ
SHELL=/bin/bash
HISTCONTROL=ignoredups
HISTSIZE=1000
HOSTNAME=testhost
GPG_TTY=/dev/pts/0
EDITOR=/usr/bin/nano
PWD=/home/silvarious/osisp-linux/lab02
```

```
LOGNAME=testuser
XDG_SESSION_TYPE=tty
CHILD_PATH=/home/silvarious/osisp-linux/lab02/out/release
…
SELINUX_LEVEL_REQUESTED=
DBUS_SESSION_BUS_ADDRESS=unix:path=/run/user/1000/bus
MAIL=/var/spool/mail/silvarious
SSH_TTY=/dev/pts/0
OLDPWD=/home/silvarious/osisp-linux/lab01/src
_=out/release/parent
Дочерний процесс завершен.
```

Ввести в терминал «q». Родительский процесс должен завершить работу, выведя сообщение о завершении.

```
q
Родительский процесс завершает работу.
```