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 №2 на тему
“Понятие процессов.”

Выполнил:
студ. гр. 350501 Везенков М. Ю.

Проверил: ст.пр. Поденок Л. П.

Минск 2025

СОДЕРЖАНИЕ

1 ЦЕЛЬ ЛАБОРАТОРНОЙ РАБОТЫ

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

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.

2.1 Описание алгоритма выполнения работы

Данная лабораторная работа включает создание двух программ: родительской (parent) и дочерней (child). Целью работы является демонстрация механизмов создания процессов, взаимодействия между ними и управления переменными окружения в операционной системе Linux. Ключевой задачей является запуск дочернего процесса с ограниченным (фильтрованным) набором переменных окружения, который определяется через конфигурационный файл.

Перед запуском родительской программы пользователь должен определить переменную окружения CHILD_PATH, значением которой является имя каталога, где находится исполняемый файл дочерней программы. При старте родительский процесс (parent) считывает свой текущий набор переменных окружения. Он сортирует эти переменные в соответствии с правилами сопоставления локали LC_COLLATE=C и выводит отсортированный список на стандартный поток вывода (stdout). Далее родительский процесс ожидает от пользователя ввода команд с клавиатуры для управления созданием дочерних процессов.

Родительская программа принимает один аргумент командной строки — путь к файлу, содержащему список имен переменных окружения, которые должны быть переданы дочернему процессу. Родительский процесс использует этот список для формирования нового, сокращенного набора переменных окружения для каждого запускаемого дочернего процесса, извлекая значения указанных переменных из своего собственного окружения.

При вводе символа + родительский процесс создает дочерний процесс с использованием системных вызовов fork() и execve(). Местоположение исполняемого файла дочерней программы определяется путем получения значения переменной CHILD_PATH с помощью функции getenv(). Имя программы для дочернего процесса (argv[0]) устанавливается в формате child_XX, где XX — это порядковый номер запуска (от 00 до 99), который инкрементируется родительским процессом при каждом запуске. Дочернему процессу передается отфильтрованный набор переменных окружения.

При вводе символа * создание дочернего процесса происходит аналогично предыдущему случаю (fork/execve, имя child_XX, отфильтрованное окружение), но информация о каталоге дочерней программы (CHILD_PATH) извлекается путем сканирования массива переменных окружения, переданного функции main() родительского процесса в качестве третьего параметра (envp).

Ввод символа & также инициирует запуск дочернего процесса (fork/execve, имя child_XX, отфильтрованное окружение). В этом случае информация о каталоге дочерней программы (CHILD_PATH) извлекается путем сканирования массива переменных окружения, доступного через внешнюю

переменную extern char **environ. Важным отличием этого режима является то, что после запуска дочернего процесса родительский процесс немедленно завершает свою работу.

В любом из режимов запуска (+, *, &), дочерний процесс (child), получив управление, сначала выводит свое имя (argv[0]), свой идентификатор процесса (PID) и идентификатор родительского процесса (PPID). Затем он определяет путь к файлу со списком переменных окружения, читая специальную переменную CHILD_ENV_FILTER_FILE из своего *полученного* окружения (эту переменную устанавливает родительский процесс). Дочерний процесс открывает этот файл, читает из него имена переменных и для каждого имени ищет соответствующее значение в своем *полученном* наборе переменных окружения. Найденные пары "ИМЯ=ЗНАЧЕНИЕ" выводятся на стандартный поток вывода. Если переменная из файла не найдена в полученном окружении, это также указывается. После вывода информации дочерний процесс завершает свое выполнение.

Ввод символа q в родительском процессе приводит к его немедленному завершению.

2.2 Описание основных функций

Программный комплекс состоит из двух исходных файлов parent.c и child.c, компилируемых в исполняемые файлы parent и child соответственно.

Функция main() в parent.c является точкой входа родительской программы. Она выполняет проверку аргументов командной строки (ожидается путь к файлу фильтра переменных окружения). Затем она выводит свой PID и отсортированный список переменных окружения, используя qsort с функцией сравнения compare_env_vars, которая применяет strcoll для учета локали LC_COLLATE=C. После этого она входит в основной цикл, считывая символы команд (+, *, &, q) от пользователя. Для команд запуска дочерних процессов вызывается функция launch_child. При получении q или после выполнения команды & цикл завершается, и программа выходит.

Функция compare_env_vars() в parent.c служит компаратором для qsort, сравнивая две строки окружения с помощью strcoll.

Функция find_env_var_value() в parent.c ищет заданную переменную окружения (var_name) в предоставленном массиве строк окружения (env_array, например, envp или environ) и возвращает указатель на ее значение или NULL, если переменная не найдена.

Функция create_filtered_env() в parent.c отвечает за создание нового массива переменных окружения для дочернего процесса. Она открывает файл фильтра (переданный как filter_filename), читает из него имена

переменных. Для каждого имени она вызывает find_env_var_value() для поиска значения в исходном окружении (source_env, обычно environ). Затем динамически выделяется память, и формируется строка "ИМЯ=ЗНАЧЕНИЕ", которая добавляется в новый массив.

Функция также добавляет запись CHILD_ENV_FILTER_FILE=путь_к_файлу_фильтра в этот массив. Возвращаемый массив завершается NULL и подходит для execve. Используется динамическая структура env_list_t для управления памятью.
Функция free_env_list() в parent.c освобождает память, выделенную для списка строк окружения функцией create_filtered_env.

Функция launch_child() в parent.c инкапсулирует логику запуска дочернего процесса. На основе символа method (+, *, &) она определяет способ получения пути к каталогу дочерней программы (CHILD_PATH) — через getenv(), сканирование main_envp или сканирование environ, используя find_env_var_value(). Затем формируется полный путь к исполняемому файлу child, имя для argv[0] (child_XX), и вызывается create_filtered_env() для подготовки окружения. Выполняется системный вызов fork(). В дочернем процессе вызывается execve() с путем к child, подготовленными argv и отфильтрованным envp. При ошибке execve дочерний процесс завершается через _exit(). В родительском процессе выводится информация о запуске, инкрементируется счетчик g_child_number, и освобождается память отфильтрованного окружения с помощью free_env_list().

Функция print_usage() в parent.c выводит справочную информацию об использовании программы при некорректном количестве аргументов.

Функция main() в child.c является точкой входа дочерней программы. Она выводит свое имя (argv[0]), PID (getpid()) и PPID (getppid()). Затем она находит путь к файлу фильтра переменных окружения, вызывая find_env_var_value_in_array() для поиска переменной CHILD_ENV_FILTER_FILE в массиве envp, переданном ей при запуске. После этого она открывает файл фильтра, читает имена переменных и для каждого имени вызывает find_env_var_value_in_array(), чтобы найти его значение *в своем полученном окружении (envp)*. Найденные пары "ИМЯ=ЗНАЧЕНИЕ" или сообщение об отсутствии переменной выводятся на stdout. В конце выводится сообщение о завершении работы.
Функция find_env_var_value_in_array() в child.c аналогична функции find_env_var_value() в parent.c, но предназначена для поиска переменных только в том массиве окружения, который был передан дочернему процессу (параметр envp функции main).

2.3 Описание порядка сборки и использования

Сборка проекта осуществляется с помощью утилиты make и предоставленного файла Makefile. Исходные коды (parent.c, child.c)

должны располагаться в каталоге src. Все результаты сборки помещаются в подкаталоги внутри каталога build.

Makefile поддерживает два основных режима сборки. Режим отладки (debug) используется по умолчанию (при вызове make без параметров или make debug-build). Компиляция выполняется с флагами отладки (-g3, -ggdb), а исполняемые файлы и вспомогательные файлы помещаются в build/debug.

Релизный режим (release) активируется командой make MODE=release или make release-build. В этом режиме применяются флаги оптимизации (-O2), а результаты сборки помещаются в build/release. Вне зависимости от режима, Makefile автоматически создает необходимый файл фильтра переменных окружения (env_vars.txt) в соответствующем каталоге сборки (build/debug или build/release). Этот файл содержит предопределенный набор имен переменных (SHELL, HOSTNAME, LOGNAME и т.д.), включая CHILD_ENV_FILTER_FILE.

Перед ручным запуском родительской программы parent необходимо экспортировать переменную окружения CHILD_PATH, указав в ней путь к каталогу, где находится скомпилированный файл child (например, export CHILD_PATH=$(pwd)/build/debug). Родительской программе также нужно передать путь к файлу env_vars.txt в качестве аргумента командной строки (например, ./build/debug/parent ./build/debug/env_vars.txt).

Для упрощения запуска Makefile предоставляет две специальные цели. Команда make run выполняет сборку в режиме отладки (если необходимо) и запускает родительскую программу, автоматически устанавливая переменную CHILD_PATH в значение $(pwd)/build/debug и передавая правильный путь к build/debug/env_vars.txt. Аналогично, команда make run-release собирает (если нужно) и запускает релизную версию, автоматически устанавливая CHILD_PATH в $(pwd)/build/release и передавая путь к build/release/env_vars.txt.

После запуска родительская программа выводит отсортированное окружение и ожидает ввода команд +, *, & или q с последующим нажатием Enter.

Для удаления всех артефактов сборки (содержимого каталога build) используется команда make clean. Это позволяет начать сборку проекта "с чистого листа".

3 РЕЗУЛЬТАТЫ РАБОТЫ ПРОГРАММЫ

```
Enter command (+, *, & to launch child, q to quit):
> +
Parent: Launching child 'child_00' using method '+'...
Parent: Child executable path: /home/sovok/My Programs/Code
Studies/ОСиСП/Везенков М.Ю./lab02/b
uild/release/child
Parent: Forked child process with PID 16635.
> Child: Name='child_00', PID=16635, PPID=16624
Child: Using environment filter file: build/debug/env_vars.txt
Child: Received Environment Variables (from filter list):
 SHELL=/bin/zsh
 HOSTNAME=(Not found in received env)
 LOGNAME=sovok
 HOME=/home/sovok
 LANG=en_US.UTF-8
 TERM=xterm-256color
 USER=sovok
 LC_COLLATE=(Not found in received env)
 PATH=/usr/local/sbin:/usr/local/bin:/usr/bin:/usr/lib/jvm/default/bin:/
usr/bin/site_perl:/usr
/bin/vendor_perl:/usr/bin/core_perl
 CHILD_ENV_FILTER_FILE=build/debug/env_vars.txt
Child: (child_00, 16635) exiting.
q
Parent: Quit command received. Exiting.
Parent: Exiting cleanly.
```

Рисунок 3.1 – пример работы команды «+»

```
Enter command (+, *, & to launch child, q to quit):
> *
Parent: Launching child 'child_00' using method '*'...
Parent: Child executable path: /home/sovok/My Programs/Code
Studies/ОСиСП/Везенков М.Ю./lab02/b
uild/release/child
Parent: Forked child process with PID 16673.
> Child: Name='child_00', PID=16673, PPID=16672
Child: Using environment filter file: build/debug/env_vars.txt
Child: Received Environment Variables (from filter list):
 SHELL=/bin/zsh
 HOSTNAME=(Not found in received env)
 LOGNAME=sovok
 HOME=/home/sovok
 LANG=en_US.UTF-8
 TERM=xterm-256color
 USER=sovok
 LC_COLLATE=(Not found in received env)
 PATH=/usr/local/sbin:/usr/local/bin:/usr/bin:/usr/lib/jvm/default/bin:/
usr/bin/site_perl:/usr
/bin/vendor_perl:/usr/bin/core_perl
 CHILD_ENV_FILTER_FILE=build/debug/env_vars.txt
Child: (child_00, 16673) exiting.
q
Parent: Quit command received. Exiting.
Parent: Exiting cleanly.
```

Рисунок 3.2 – пример работы команды «*»

```
Enter command (+, *, & to launch child, q to quit):
> &
Parent: Launching child 'child_00' using method '&'...
Parent: Child executable path: /home/sovok/My Programs/Code
Studies/ОСиСП/Везенков М.Ю./lab02/b
uild/release/child
Parent: Forked child process with PID 16699.
Parent: Initiating termination after launching child via '&'.
Parent: Exiting cleanly.
Child: Name='child_00', PID=16699, PPID=1058
Child: Using environment filter file: build/debug/env_vars.txt
Child: Received Environment Variables (from filter list):
 SHELL=/bin/zsh
 HOSTNAME=(Not found in received env)
 LOGNAME=sovok
 HOME=/home/sovok
 LANG=en_US.UTF-8
 TERM=xterm-256color
 USER=sovok
 LC_COLLATE=(Not found in received env)
 PATH=/usr/local/sbin:/usr/local/bin:/usr/bin:/usr/lib/jvm/default/bin:/
usr/bin/site_perl:/usr
/bin/vendor_perl:/usr/bin/core_perl
 CHILD_ENV_FILTER_FILE=build/debug/env_vars.txt
Child: (child_00, 16699) exiting.
```

Рисунок 3.3 – пример работы команды «&»

4 ВЫВОД

При выполнении лабораторной работы были изучены принципы управления процессами и их окружением в операционной системе `Linux`.

В ходе выполнения работы был разработан программный комплекс (родительский и дочерний процессы), который демонстрирует создание новых процессов с помощью `fork` и `execve`, модификацию и передачу переменных окружения между ними, а также запуск дочерних процессов в различных режимах доступа к окружению и в фоновом режиме.