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

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

Родительский процесс, запуская дочерний, создает для него сокращенную среду (окружение). Для этого пользователем создается файл env, содержащий небольшой набор имен переменных окружения, передаваемых при вызове execve().

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

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, установленные оболочкой, 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.

### Символ «+»

Родительский процесс, 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 дочерним процессом для получения ему переданных 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 «q»**

Завершает выполнение родительского процесса после завершения дочернего.

## 2 ВЫПОЛНЕНИЕ ЛАБОРАТОРНОЙ РАБОТЫ

### 2.1 Описание алгоритма выполнения работы

Данная работа включает разработку двух программ: родительской (parent) и дочерней (child), демонстрирующих взаимодействие процессов и управление окружением в Linux. Основная цель — запуск дочернего процесса с ограниченным набором переменных окружения, определяемым пользователем.

При запуске родительский процесс сначала получает и выводит на стандартный вывод (stdout) свой собственный набор переменных окружения, предварительно отсортировав их в соответствии с правилами локали LC_COLLATE=C. Затем он считывает специальный файл env, созданный пользователем, который содержит список имен переменных окружения для передачи дочернему процессу. Родительский процесс использует значения этих переменных из *своего* окружения для формирования нового, сокращенного окружения для дочернего процесса. Для определения местоположения исполняемого файла дочерней программы (child), родительский процесс использует переменную окружения CHILD_PATH, которую пользователь должен установить перед запуском parent.

После начальной настройки родительский процесс входит в цикл обработки команд пользователя, вводимых с клавиатуры. При вводе символа +, родительский процесс создает новый дочерний процесс с использованием системных вызовов fork(2) и execve(2). Имя дочерней программы (argv[0]) устанавливается в формате child_XX, где XX — порядковый номер (00-99), инкрементируемый родителем. Вторым параметром (argv[1]) дочернему процессу передается путь к файлу env. Дочерний процесс (child), в свою очередь, выводит свое имя (argv[0]), PID и PPID, открывает переданный ему файл env, читает из него имена переменных и, используя системный вызов getenv(), получает их значения из *своего* (сокращенного) окружения и выводит их на stdout. Родитель кратковременно приостанавливается перед выводом следующего приглашения.

Если пользователь вводит символ *, запуск дочернего процесса происходит аналогично предыдущему случаю (fork/execve, имя child_XX, сокращенное окружение), однако дочернему процессу не передается путь к файлу env. Вместо этого, дочерний процесс получает информацию о своем окружении путем сканирования массива строк, переданного в качестве третьего параметра функции main() (envp[]), и выводит его содержимое на stdout; родитель также кратковременно приостанавливается.

Ввод символа & инициирует запуск дочернего процесса аналогично режиму + (с именем child_XX, сокращенным окружением и передачей пути к env в argv[1]), но с ключевым отличием: родительский процесс не ожидает

завершения или приостановки дочернего процесса, а немедленно возвращает пользователю приглашение для ввода следующей команды, позволяя дочернему процессу выполняться в фоновом режиме. Наконец, ввод символа q завершает выполнение родительского процесса; любые фоновые дочерние процессы, которые могут все еще выполняться, станут "сиротами". Программа обеспечивает наглядную демонстрацию создания процессов, передачи и модификации окружения, а также базовое управление выполнением дочерних процессов в переднем и фоновом планах.

### 2.2 Описание основных функций

Программный комплекс состоит из двух исполняемых файлов (parent и child), компилируемых из соответствующих исходных кодов (parent.c и child.c). Ключевые функции распределены между ними для выполнения поставленных задач.

Функция main() в parent.c является точкой входа для родительской программы. Она отвечает за ряд действий: получает путь к каталогу дочерней программы из переменной окружения CHILD_PATH, формирует полные пути к исполняемому файлу child и файлу окружения env, проверяет их существование и права доступа, вызывает print_sorted_environ() для вывода отсортированного окружения родителя, реализует основной цикл обработки команд пользователя (+, *, &, q), вызывает launch_child() для запуска дочерних процессов и обеспечивает корректное завершение работы.

Функция print_sorted_environ() в parent.c получает доступ к глобальному массиву environ, копирует указатели на строки окружения, устанавливает локаль LC_COLLATE в "C" с помощью setlocale(), сортирует скопированные указатели с использованием qsort() и функции сравнения compare_strings() (которая использует strcmp), после чего выводит отсортированный список переменных окружения родителя.

Функция build_child_env() в parent.c открывает и читает файл env. Для каждого имени переменной из файла она вызывает getenv() для получения ее значения из окружения родителя. Далее динамически выделяется память и формируется массив строк char ** в формате "ИМЯ=ЗНАЧЕНИЕ", завершенный NULL-указателем, который пригоден для передачи в execve(). Функция также обрабатывает ситуацию, когда переменная из env не найдена в окружении родителя.

Функция launch_child() в parent.c является основной для запуска дочернего процесса. Она проверяет лимит на количество дочерних процессов, вызывает build_child_env() для подготовки массива окружения, формирует имя дочернего процесса (child_XX) и массив аргументов argv для execve. Затем используется fork() для создания нового процесса. В дочернем процессе вызывается execve(), передавая путь к child, подготовленные argv и envp; при ошибке execve дочерний процесс завершается через _exit(). В

родительском процессе выводится информация о запуске, инкрементируется счетчик child_counter, освобождается память из-под child_envp с помощью free_child_env(). В зависимости от флага is_background, родитель либо немедленно возвращает управление (для &), либо делает короткую паузу с usleep() (для + и *).

Функция main() в child.c служит точкой входа дочерней программы. Она выводит свое имя (argv[0]), PID (getpid()) и PPID (getppid()), а затем определяет режим работы (+ или *) на основе количества аргументов argc. Если режим +, дочерний процесс открывает файл env (путь из argv[1]), читает имена переменных, вызывает getenv() для получения их значений из *своего* окружения и выводит результат. Если же режим *, он итерирует по массиву envp, переданному как третий аргумент main(), и выводит каждую строку "ИМЯ=ЗНАЧЕНИЕ".

В конце выводится сообщение о завершении работы.
К вспомогательным функциям относятся compare_strings() для qsort и free_child_env() для освобождения памяти в parent.c.

### 2.3 Описание порядка сборки и использования

Для сборки проекта используется утилита make и предоставленный Makefile. Структура сборки предполагает размещение исходных кодов в каталоге src, а всех артефактов сборки — в подкаталогах каталога build. Поддерживается два режима сборки.

Первый — режим отладки (debug), который используется по умолчанию при вызове make или make debug-build. Компиляция в этом режиме происходит с флагами, облегчающими отладку (-g2, -ggdb), а результаты помещаются в каталог build/debug.

Второй — релизный режим (release), активируемый командой make MODE=release или make release-build. В этом случае используются флаги оптимизации (-O2) и, возможно, более строгие флаги проверки, а результаты помещаются в каталог build/release. В обоих режимах

Makefile автоматически создает файл env с минимально необходимым набором переменных в соответствующем каталоге (build/debug или build/release).

Перед запуском родительской программы необходимо убедиться, что переменная окружения CHILD_PATH установлена и указывает на каталог, содержащий *скомпилированный* исполняемый файл child. Для удобства запуска Makefile предоставляет специальные цели: make run собирает и запускает отладочную версию, автоматически устанавливая CHILD_PATH=build/debug, а make run-release делает то же самое для релизной версии, устанавливая CHILD_PATH=build/release.

Во время выполнения родительская программа ожидает ввода команд +, *, & или q в консоли.

Для очистки всех созданных в процессе сборки файлов и каталогов (build/*) используется команда make clean, что позволяет выполнить "чистую" сборку проекта с нуля.

## 3 РЕЗУЛЬТАТЫ РАБОТЫ ПРОГРАММЫ

```
Enter command (+ = launch fg getenv, * = launch fg envp, & = launch bg getenv, q
= quit):
> +
Command: +
Parent: Launched child 'child_00' (PID: 11281).
Child Process Report:
 Name (argv[0]): child_00
 PID: 11281
 PPID: 11261
Child Environment Variables:
 Mode: '+' (using getenv based on build/debug/env)
   SHELL=/bin/zsh
   HOME=/home/sovok
   HOSTNAME (not found via getenv)
   LOGNAME=sovok
   LANG=en_US.UTF-8
   TERM=xterm-256color
   USER=sovok
   LC_COLLATE (not found via getenv)
   PATH=/usr/local/sbin:/usr/local/bin:/usr/bin:/usr/lib/jvm/default/bin:/usr/
bin/site_perl:/u
sr/bin/vendor_perl:/usr/bin/core_perl
Child process finished.
```

Рисунок 3.1 – пример работы команды «+»

```
> *
Command: *
Parent: Info: Variable 'HOSTNAME' from env not found in parent's environment, not
passed to chi
ld in '*' mode.
Parent: Info: Variable 'LC_COLLATE' from env not found in parent's environment,
not passed to c
hild in '*' mode.
Parent: Launched child 'child_01' (PID: 11282).
Child Process Report:
 Name (argv[0]): child_01
 PID: 11282
 PPID: 11261
Child Environment Variables:
 Mode: '*' (iterating envp[] parameter)
   SHELL=/bin/zsh
   HOME=/home/sovok
   LOGNAME=sovok
   LANG=en_US.UTF-8
   TERM=xterm-256color
   USER=sovok
   PATH=/usr/local/sbin:/usr/local/bin:/usr/bin:/usr/lib/jvm/default/bin:/usr/
bin/site_perl:/u
sr/bin/vendor_perl:/usr/bin/core_perl
Child process finished.
```

Рисунок 3.2 – пример работы команды «*»

```
> &
Command: & (Launching child in background using '+' mode)
Parent: Launched child 'child_02' (PID: 11285) in background.
> Child Process Report:
 Name (argv[0]): child_02
 PID: 11285
 PPID: 11261
Child Environment Variables:
 Mode: '+' (using getenv based on build/debug/env)
   SHELL=/bin/zsh
   HOME=/home/sovok
   HOSTNAME (not found via getenv)
   LOGNAME=sovok
   LANG=en_US.UTF-8
   TERM=xterm-256color
   USER=sovok
   LC_COLLATE (not found via getenv)
   PATH=/usr/local/sbin:/usr/local/bin:/usr/bin:/usr/lib/jvm/default/bin:/usr/
bin/site_perl:/u
sr/bin/vendor_perl:/usr/bin/core_perl
Child process finished.
```

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.