БГУИР

Кафедра ЭВМ

Операционные системы и системное программирование
Отчет по лабораторной работе № 2
Тема: «Понятие процессов»

Выполнил:
студент группы 230501 Кочеров Р.С.

Проверил:
Поденок Л.П.

Минск
2024

1 УСЛОВИЕ ЛАБОРАТОРНОЙ РАБОТЫ.

Разработать 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 «+», используя fork(2) и execve(2) порождает дочерний процесс и запускает в нем очередной экземпляр программы child. Информацию о каталоге, где размещается child, получает из окружения, 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 расположении программы child получает, сканируя массив параметров среды, переданный в третьем параметре функции main().

Символ «&» порождает дочерний процесс аналогично предыдущему случаю, однако информацию о расположении программы child получает, сканируя массив параметров среды, указанный 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, HOME, HOSTNAME, LOGNAME, 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 «q» завершает выполнение родительского процесса.

2 ОПИСАНИЕ АЛГОРИТМОВ И РЕШЕНИЙ.

Программа предназначена для запуска родительского процесса `parent`, процесс выводит свое окружение и переходит в обработку символов, обработка включает три случая «&», «+», «*» - она берет тремя способами среду окружения `CHILD_PATH`, в которой должно хранится путь к дочерней программе, далее запуская дочерний процесс и он выводит свои данные.

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

2.1. Программа parent

Данную программу мы запускаем, для того чтобы запустить дочерний процесс. Данный код выполняет следующие действия:

1. Включает необходимые заголовочные файлы.

2. Определяет максимальное количество переменных окружения (MAX_ENV_VARS).

3. Объявляет прототипы функций: compare_strings, sort_and_print_env_vars, do_child_number, strdup.

4. Объявляет глобальную переменную environ, которая представляет собой массив указателей на переменные окружения.

5. Определяет функцию main, которая является точкой входа в программу.

Функция compare_strings сравнивает две строки с учетом текущей локали (LC_COLLATE).

Функция sort_and_print_env_vars получает переменные окружения, копирует их в массив, сортирует массив и выводит отсортированные переменные окружения на стандартный вывод.

Функция do_child_number создает строку, содержащую имя дочернего процесса в формате "child_XX", где XX - число, переданное в качестве аргумента функции.

В функции main происходит следующее:

- Устанавливается локаль LC_COLLATE в значение "C".

- Вызывается функция sort_and_print_env_vars для сортировки и вывода переменных окружения.

- Открывается файл "param.txt" для чтения.

- Читаются названия переменных из файла, значения которых затем получаются из среды.

- Если переменная найдена, создается строка в формате "название=значение" и добавляется в массив env_vars.

- Если превышено максимальное количество переменных окружения (MAX_ENV_VARS), выводится сообщение об ошибке.

- Закрывается файл "param.txt".

- Запускается цикл, в котором пользователю предлагается выбрать опцию.

- В зависимости от выбранной опции выполняются следующие действия:

- Опция '+' ищет переменную окружения CHILD_PATH в среде и выводит ее значение, если оно установлено.

- Опция '*' ищет переменную окружения CHILD_PATH в массиве параметров envp и выводит ее значение, если оно установлено.

- Опция '&' ищет переменную окружения CHILD_PATH в массиве параметров environ и выводит ее значение, если оно установлено.

- Опция 'q' завершает программу.

- При других недопустимых входных данных выводится сообщение об ошибке.

- Если выбрана опция '+' или '*' или '&', создается массив аргументов args для запуска дочернего процесса.

- Вызывается функция fork для создания дочернего процесса.

- В дочернем процессе вызывается функция execve для запуска программы с путем, указанным в переменной childPath, и передачей массива аргументов args и переменных окружения env_vars.

- Родительский процесс ожидает завершения дочернего процесса и выводит его статус завершения.

- Цикл повторяется, пока выбранная опция не будет 'q'.

- По завершении программы освобождаются выделенные ресурсы и процесс завершается.

2.2. Программа child

Дочерняя программа `child` получает из `parent` имя программы с определенным номером, название файла, в котором хранятся определенные ключи из окружения, и символ который был обработан в `parent`.

Данный код является программой на языке C, которая выполняет следующие действия:

1. Подключает необходимые заголовочные файлы.
2. Выводит сообщение "Child process begins..." в стандартный поток вывода.
3. Объявляет переменные pid и ppid типа pid_t для хранения идентификаторов текущего процесса и родительского процесса соответственно.
4. Устанавливает локаль для сравнения строк в стандартной локали "C".
5. Получает идентификатор текущего процесса с помощью функции getpid() и идентификатор родительского процесса с помощью функции getppid().
6. Выводит название программы (имя исполняемого файла) и идентификаторы текущего процесса и родительского процесса в стандартный поток вывода.
7. Создает массив указателей на строки env_vars, размер которого ограничен константой MAX_ENV_VARS.
8. Открывает файл "param.txt" для чтения.
9. Если открытие файла завершилось неудачей, выводит сообщение об ошибке и завершает программу с кодом возврата 1.
10. Читает строки из файла "param.txt" и помещает их в массив env_vars.
11. Для каждой строки, прочитанной из файла, удаляет символ новой строки, получает значение соответствующей переменной окружения с помощью функции getenv() и выводит в стандартный поток вывода пару "название переменной = значение переменной".
12. Увеличивает счетчик var_count для отслеживания количества прочитанных переменных.
13. Если превышено максимальное количество переменных окружения MAX_ENV_VARS, выводит сообщение "Превышено максимальное количество переменных окружения" и прекращает чтение из файла.
14. Устанавливает NULL-терминатор в массиве env_vars.
15. Закрывает файл "param.txt".
16. Завершает программу с кодом возврата 0.

3. ФУНКЦИОНАЛЬНАЯ СТРУКТУРА ПРОЕКТА.

Проект собирается с помощью makefile. Перед запуском проекта требуется создать переменную среды CHILD_PATH, где будет хранится имя дочерней программы. Пример создания среды переменной:

```
rlinux@fedora:~$export CHILD_PATH="build/debug/child"
```

Для запуска проекта нам требуется в терминале заупустить программу parent. Где выводит все переменные среды включающая CHILD_PATH и переходит в обработку символов где и запускает child.

В проекте имеется каталог для сборки debug и release. Каталог git для системы контроля версий моего проекта. Директория src с исходным кодом. И makefile для компиляции и сборки моего проекта.

4. ПОРЯДОК СБОРКИ И ИСПОЛЬЗОВАНИЯ.

Для компиляции и сборки проекта используется makefile.

Порядок сборки:

1) Задание переменных:

- `DEBUG и RELEASE` - пути к каталогам для отладочной и релизной сборки соответственно.

- `OUT_DIR` - текущий каталог для выходных файлов (по умолчанию используется отладочная сборка).

- `FLAGS_DEBUG и CFLAGS_RELEASE` - флаги компиляции для отладочной и релизной сборки соответственно.

`object_child` и `object_parent` - объектные файлы для компиляции.

`Parent и child` - имя выходного исполняемого файла.

2. Определение компилятора и флагов компиляции:

`CC` - компилятор (gcc).

`CFLAGS` - флаги компиляции, выбираются в зависимости от переменной `MODE` (отладочная или релизная сборка).

3. Определение зависимостей:

`vpath` - указание директорий для поиска файлов с исходным кодом и заголовочных файлов.

`ifeq ($(MODE), release)` - установка флагов и каталогов в случае релизной сборки.

4. Определение целей:

`all` - основная цель, компиляция всех объектных файлов и создание исполняемого файла.

`$(child) и $(parent)` - правило для создания исполняемого файла.

`$(OUT_DIR)/%.o: %.c` - правило для компиляции каждого исходного файла в объектный.

Порядок использования.

1. Компиляция:

Для отладочной сборки: `make` или `make MODE=debug`

Для релизной сборки: `make MODE=release`

2. Очистка:

`make clean` - удаляет все объектные файлы и исполняемый файл.

3. Запуск:

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

5. МЕТОД ТЕСТИРОВАНИЯ И РЕЗУЛЬТАТ ТЕСТИРОВАНИЯ.

1. Создание CHILD_PATH:

```
rlinux@fedora:~$export CHILD_PATH="build/debug/child"
```

2. Запуск `parent`:

rlinux@fedora:~/Kocherov/lab02$ export CHILD_PATH=./build/debug/child

rlinux@fedora:~/Kocherov/lab02$ build/debug/parent

CHILD_PATH=./build/debug/child

COLORFGBG=15;0

COLORTERM=truecolor

CVS_RSH=ssh

DBUS_SESSION_BUS_ADDRESS=unix:path=/run/user/1000/bus

DEBUGINFOD_URLS=https://debuginfod.fedoraproject.org/

DESKTOP_SESSION=plasma

DISPLAY=:0

EDITOR=/usr/bin/nano

GDK_CORE_DEVICE_EVENTS=1

GTK2_RC_FILES=/home/rlinux/.gtkrc-2.0-kde4

GTK_RC_FILES=/etc/gtk/gtkrc:/home/rlinux/.gtkrc:/home/rlinux/.config/gtkrc

HISTCONTROL=ignoredups

HISTSIZE=1000

HOME=/home/rlinux

HOSTNAME=fedora

IMSETTINGS_INTEGRATE_DESKTOP=yes

IMSETTINGS_MODULE=X compose table

INVOCATION_ID=3e1f164fd39043139adb0f391ffbc0dc

JOURNAL_STREAM=8:13606

KATE_PID=3410

KDEDIRS=/usr

KDE_APPLICATIONS_AS_SCOPE=1

KDE_FULL_SESSION=true

KDE_SESSION_UID=1000

KDE_SESSION_VERSION=5

KONSOLE_DBUS_SERVICE=:1.93

KONSOLE_DBUS_SESSION=/Sessions/1

KONSOLE_VERSION=230805

LANG=ru_RU.UTF-8

LANGUAGE=

LESSOPEN=||/usr/bin/lesspipe.sh %s

LOGNAME=rlinux

LS_COLORS= null

MAIL=/var/spool/mail/rlinux

MANAGERPID=1307

MEMORY_PRESSURE_WATCH=/sys/fs/cgroup/user.slice/user-1000.slice/user@1000.service/session.slice/plasma-plasmashell.service/memory.pressure

MEMORY_PRESSURE_WRITE=c29tZSAyMDAwMDAgMjAwMDAwMAA=
OLDPWD=/home/rlinux/Kocherov/lab02/src
PAM_KWALLET5_LOGIN=/run/user/1000/kwallet5.socket
PATH=/home/rlinux/.local/bin:/home/rlinux/bin:/usr/local/bin:/usr/bin:/bin:/usr/local/sbin:/usr/sbin:/sbin
PLASMA_USE_QT_SCALING=1
PROFILEHOME=
PWD=/home/rlinux/Kocherov/lab02
QT_AUTO_SCREEN_SCALE_FACTOR=0
QT_IM_MODULE=xim
QT_WAYLAND_DECORATION=adwaita
QT_WAYLAND_FORCE_DPI=96
SESSION_MANAGER=local/unix:@/tmp/.ICE-unix/1663,unix/unix:/tmp/.ICE-unix/1663
SHELL=/bin/bash
SHELL_SESSION_ID=ffc31430a9d647f5ac32e7391dccc7da
SHLVL=1
SSH_ASKPASS=/usr/bin/ksshaskpass
SSH_AUTH_SOCK=/run/user/1000/ssh-agent.socket
SYSTEMD_EXEC_PID=1714
TERM=xterm-256color
USER=rlinux
WAYLAND_DISPLAY=wayland-0
WINDOWID=0
XAUTHORITY=/run/user/1000/xauth_ZNxiDv
XCURSOR_SIZE=24
XCURSOR_THEME=breeze_cursors
XDG_ACTIVATION_TOKEN=kwin-1
XDG_CONFIG_DIRS=/home/rlinux/.config/kdedefaults:/etc/xdg:/usr/share/kde-settings/kde-profile/default/xdg
XDG_CURRENT_DESKTOP=KDE
XDG_DATA_DIRS=/home/rlinux/.local/share/flatpak/exports/share:/var/lib/flatpak/exports/share:/usr/local/share:/usr/share
XDG_MENU_PREFIX=kf5-
XDG_RUNTIME_DIR=/run/user/1000
XDG_SEAT=seat0
XDG_SEAT_PATH=/org/freedesktop/DisplayManager/Seat0
XDG_SESSION_CLASS=user
XDG_SESSION_DESKTOP=KDE
XDG_SESSION_ID=2
XDG_SESSION_PATH=/org/freedesktop/DisplayManager/Session1
XDG_SESSION_TYPE=wayland
XDG_VTNR=2

XKB_DEFAULT_LAYOUT=us,ru
XKB_DEFAULT_MODEL=pc105
XKB_DEFAULT_OPTIONS=grp:alt_shift_toggle
XKB_DEFAULT_VARIANT=,
XMODIFIERS=@im=none
_=build/debug/parent
Переменная окружения LC_COLLATE не найдена
Your choice:
[+]
[*]
[&]
[q]
+
CHILD_PATH+: ./build/debug/child
Child process created. Please, wait...
Child process begins...
My name: child_01
My pid = 3497, my ppid = 3496
SHELL = /bin/bash
HOME = /home/rlinux
HOSTNAME = fedora
LOGNAME = (null)
LANG = ru_RU.UTF-8
TERM = xterm-256color
USER = rlinux
LC_COLLATE = (null)
PATH = /home/rlinux/.local/bin:/home/rlinux/bin:/usr/local/bin:/usr/bin:/bin:/usr/local/sbin:/usr/sbin:/sbin
Child process have ended with 0 exit status
Your choice:
[+]
[*]
[&]
[q]
*
CHILD_PATH*: ./build/debug/child
Child process created. Please, wait...
Child process begins...
My name: child_02
My pid = 3498, my ppid = 3496
SHELL = /bin/bash
HOME = /home/rlinux
HOSTNAME = fedora

```
        LOGNAME = (null)
        LANG = ru_RU.UTF-8
        TERM = xterm-256color
        USER = rlinux
        LC_COLLATE = (null)
        PATH =
/home/rlinux/.local/bin:/home/rlinux/bin:/usr/local/bin:/usr/bin:/bin:/usr/local
/sbin:/usr/sbin:/sbin
        Child process have ended with 0 exit status
        Your choice:
        [+]
        [*]
        [&]
        [q]
        &
        CHILD_PATH&: ./build/debug/child
        Child process created. Please, wait...
        Child process begins...
        My name: child_03
        My pid = 3499, my ppid = 3496
        SHELL = /bin/bash
        HOME = /home/rlinux
        HOSTNAME = fedora
        LOGNAME = (null)
        LANG = ru_RU.UTF-8
        TERM = xterm-256color
        USER = rlinux
        LC_COLLATE = (null)
        PATH =
/home/rlinux/.local/bin:/home/rlinux/bin:/usr/local/bin:/usr/bin:/bin:/usr/local
/sbin:/usr/sbin:/sbin
        Child process have ended with 0 exit status
        Your choice:
        [+]
        [*]
        [&]
        [q]
        q
exiting program
```