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 №2
на тему
Понятие процессов.

Выполнил студент гр.230501 Лазовский И.А.

Проверил старший преподаватель кафедры ЭВМ
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.

- `environment_output(char *envp[])`: Данная функция выводит все переменные окружения, переданные в качестве аргумента `envp[]`. Она перебирает элементы массива `envp[]` и выводит их на экран.

- `compare_env(const void *a, const void *b)`: Эта функция используется в функции `qsort()` для сравнения двух элементов массива переменных окружения. Она возвращает результат сравнения двух строк.

- `main(int argc, char *argv[], char *envp[])`: Это основная функция программы. В начале она выполняет сортировку переменных окружения с помощью функции `qsort()` и выводит отсортированный список с помощью функции `environment_output()`. Затем она входит в бесконечный цикл, в котором ожидает ввод символа для управления.

- Если введен символ q или достигнут лимит на количество дочерних программ (100), программа выводит сообщение о выходе из родительского окружения и завершает свою работу.

- Если введены символы *, & или +, программа создает новый дочерний процесс с помощью функции `fork()`. В дочернем процессе выполняется код, в котором определяется путь к дочерней программе и запускается с помощью функции `execve()`. В родительском процессе увеличивается счетчик дочерних программ, и процесс ожидает завершения дочернего процесса с помощью функции `wait(NULL)`.

2.2. Программа child

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

- `main(int argc, char *argv[], char *envp[])` - в этой функции функции `main` он получает идентификаторы текущего процесса (pid) и родительского процесса (ppid) с помощью функций `getpid()` и `getppid()` соответственно. Затем он выводит информацию о текущем процессе, включая его имя, `pid` и `ppid`. После этого он открывает файл, имя которого передается в качестве первого аргумента командной строки (`argv[1]`), и читает его построчно. Для каждой прочитанной строки он удаляет символ новой строки с помощью функции `strcspn()`, а затем выводит эту строку и соответствующую ей переменную окружения,

полученную с помощью функции `getenv()`. После чтения всех строк файла он закрывает файл и завершает выполнение программы с кодом выхода 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:

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

Для запуска проекта нам требуется в терминале запустить программу parent. Которая 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, например: ./build/debug/parent.

5. МЕТОД ТЕСТИРОВАНИЯ И РЕЗУЛЬТАТ ТЕСТИРОВАНИЯ.

1. Создание CHILD_PATH:

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

2. Запуск parent:

```
ilua@fedora:~/Рабочий стол/labs/project/test_for_lab2#
build/debug/parent vrop.txt
PARENT ENVIRONMENT OUTPUT:
CHILD_PATH=build/debug/child
COLORFGBG=15;0
COLORTERM=truecolor
DBUS_SESSION_BUS_ADDRESS=unix:path=/run/user/0/bus
DEBUGINFOD_URLS=https://debuginfod.fedoraproject.org/
DESKTOP_SESSION=plasma
DISPLAY=:0
EDITOR=/usr/bin/nano
GDK_CORE_DEVICE_EVENTS=1
GTK2_RC_FILES=/root/.gtkrc-2.0-kde4
HISTCONTROL=ignoreboth
HISTSIZE=1000
HOME=/root
HOSTNAME=fedora
IMSETTINGS_INTEGRATE_DESKTOP=yes
IMSETTINGS_MODULE=X compose table
INVOCATION_ID=4dc6677e3fe141a28adf2ef8cff5ef14
JOURNAL_STREAM=8:18370
KDEDIRS=/usr
KDE_APPLICATIONS_AS_SCOPE=1
KDE_FULL_SESSION=true
KDE_SESSION_UID=0
```

```
KDE_SESSION_VERSION=5
KGLOBALACCELD_PLATFORM=org.kde.kwin
KONSOLE_DBUS_SERVICE=:1.91
KONSOLE_DBUS_SESSION=/Sessions/1
KONSOLE_DBUS_WINDOW=/Windows/1
KONSOLE_VERSION=230805
LANG=ru_RU.UTF-8
LANGUAGE=
LESSOPEN=||/usr/bin/lesspipe.sh %s
LOGNAME=root
LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35:do=
01;35:bd=40;33;01:cd=40;33;01:or=40;31;01:mi=01;37;41:su=37;
41:sg=30;43:ca=00:tw=30;42:ow=34;42:st=37;44:ex=01;32:*.tar=
01;31:*.tgz=01;31:*.arc=01;31:*.arj=01;31:*.t
az=01;31:*.lha=01;31:*.lz4=01;31:*.lzh=01;31:*.lzma=01;31:*.
tlz=01;31:*.txz=01;31:*.tzo=01;31:*.t7z=01;31:*.zip=01;31:*.
z=01;31:*.dz=01;31:*.gz=01;31:*.lrz=01;31:*.lz=01;31:*.lzo=0
1;31:*.xz=01;31:*.zst=01;31:*.tzst=01;31:*.bz
2=01;31:*.bz=01;31:*.tbz=01;31:*.tbz2=01;31:*.tz=01;31:*.deb
=01;31:*.rpm=01;31:*.jar=01;31:*.war=01;31:*.ear=01;31:*.sar
=01;31:*.rar=01;31:*.alz=01;31:*.ace=01;31:*.zoo=01;31:*.cpi
o=01;31:*.7z=01;31:*.rz=01;31:*.cab=01;31:*.w
im=01;31:*.swm=01;31:*.dwm=01;31:*.esd=01;31:*.avif=01;35:*.
jpg=01;35:*.jpeg=01;35:*.mjpg=01;35:*.mjpeg=01;35:*.gif=01;3
5:*.bmp=01;35:*.pbm=01;35:*.pgm=01;35:*.ppm=01;35:*.tga=01;3
5:*.xbm=01;35:*.xpm=01;35:*.tif=01;35:*.tiff=
01;35:*.png=01;35:*.svg=01;35:*.svgz=01;35:*.mng=01;35:*.pcx
=01;35:*.mov=01;35:*.mpg=01;35:*.mpeg=01;35:*.m2v=01;35:*.mk
v=01;35:*.webm=01;35:*.webp=01;35:*.ogm=01;35:*.mp4=01;35:*.
m4v=01;35:*.mp4v=01;35:*.vob=01;35:*.qt=01;35
:*.nuv=01;35:*.wmv=01;35:*.asf=01;35:*.rm=01;35:*.rmvb=01;35
:*.flc=01;35:*.avi=01;35:*.fli=01;35:*.flv=01;35:*.gl=01;35:
*.dl=01;35:*.xcf=01;35:*.xwd=01;35:*.yuv=01;35:*.cgm=01;35:*
.emf=01;35:*.ogv=01;35:*.ogx=01;35:*.aac=01;3
6:*.au=01;36:*.flac=01;36:*.m4a=01;36:*.mid=01;36:*.midi=01;
36:*.mka=01;36:*.mp3=01;36:*.mpc=01;36:*.ogg=01;36:*.ra=01;3
6:*.wav=01;36:*.oga=01;36:*.opus=01;36:*.spx=01;36:*.xspf=01
;36:*~=00;90:*#=00;90:*.bak=00;90:*.old=00;90
:*.orig=00;90:*.part=00;90:*.rej=00;90:*.swp=00;90:*.tmp=00;
90:*.dpkg-dist=00;90:*.dpkg-old=00;90:*.ucf-
dist=00;90:*.ucf-new=00;90:*.ucf-
old=00;90:*.rpmnew=00;90:*.rpmorig=00;90:*.rpmsave=00;90:
MAIL=/var/spool/mail/root
MANAGERPID=1412
MC_SID=3688
MC_TMPDIR=/var/tmp/mc-root
MEMORY_PRESSURE_WATCH=/sys/fs/cgroup/user.slice/user-
```

```
0.slice/user@0.service/session.slice/plasma-
kwin_wayland.service/memory.pressure
MEMORY_PRESSURE_WRITE=c29tZSAyMDAwMDAgMjAwMDAwMAA=
MY_VARIABLE=my_value
OLDPWD=/root/Рабочий стол/labs/project
PATH=/root/.local/bin:/root/bin:/usr/local/bin:/usr/bin:/bin
:/usr/local/sbin:/usr/sbin:/sbin
PLASMA_USE_QT_SCALING=1
PROFILEHOME=
PWD=/root/Рабочий стол/labs/project/test_for_lab2
QSG_RENDER_LOOP=basic
QT_AUTO_SCREEN_SCALE_FACTOR=0
QT_IM_MODULE=xim
QT_WAYLAND_DECORATION=adwaita
QT_WAYLAND_FORCE_DPI=96
SHELL=/bin/bash
SHELL_SESSION_ID=7298863a77da465a8000b9a5b663faaa
SHLVL=2
SSH_ASKPASS=/usr/bin/ksshaskpass
SSH_AUTH_SOCK=/run/user/0/ssh-agent.socket
SYSTEMD_EXEC_PID=1614
TERM=xterm-256color
USER=root
WAYLAND_DISPLAY=wayland-0
WINDOWID=1
XAUTHORITY=/run/user/0/xauth_crgXGh
XCURSOR_SIZE=24
XCURSOR_THEME=breeze_cursors
XDG_ACTIVATION_TOKEN=kwin-6
XDG_CONFIG_DIRS=/root/.config/kdedefaults:/etc/xdg:/usr/shar
e/kde-settings/kde-profile/default/xdg
XDG_CURRENT_DESKTOP=KDE
XDG_DATA_DIRS=/root/.local/share/flatpak/exports/share:/var/
lib/flatpak/exports/share:/usr/local/share:/usr/share
XDG_MENU_PREFIX=kf5-
XDG_RUNTIME_DIR=/run/user/0
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
Введите символ для управления:+
I child program. My name = child_00, my pid = 4063, my ppid
= 4062
CHILD ENVIRONMENT:
Введите символ для управления:*
I child program. My name = child_01, my pid = 4064, my ppid
= 4062
CHILD ENVIRONMENT:
Введите символ для управления:&
I child program. My name = child_02, my pid = 4065, my ppid
= 4062
CHILD ENVIRONMENT:
Введите символ для управления:q
Выход из родительского окружения.
```