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

Лабораторная работа №2
«Понятие процессов»

Выполнил:
Студент группы 350501
Рутковский В.К.

Проверил:
Поденок Л.П.

Минск 2025

1 ПОСТАНОВКА ЗАДАЧИ

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 «+», 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

Символ «q» завершает выполнение родительского процесса после завершения дочернего.

2 ОПИСАНИЕ РАБОТЫ ПРОГРАММЫ

2.1 Описание работы программы

Эта программа управляет созданием дочерних процессов и передачей им переменных окружения разными способами. Родительский процесс (main в файле parent.c) загружает переменные окружения из файла, считывает путь

к дочернему процессу из переменной CHILD_PATH и обрабатывает пользовательский ввод, позволяя создавать дочерние процессы с разными параметрами. В зависимости от нажатой клавиши (+, *, &), новые процессы запускаются с окружением из файла, массива переменных или стандартного окружения. Родительский процесс также отслеживает завершение дочерних процессов и освобождает выделенную память перед выходом.

Дочерний процесс (main в файле child.c) получает от родителя имя исполняемого файла и информацию о переменных окружения. В зависимости от переданных аргументов он либо загружает и выводит переменные из файла, либо отображает переменные, переданные через envp. Это позволяет проверить, каким образом окружение передаётся в зависимости от режима запуска. В целом программа демонстрирует работу с процессами, передачу переменных окружения и обработку пользовательского ввода для управления дочерними процессами.

2.2 Описание основных функций

Программа состоит из нескольких подпрограмм (частей программы), представляющих собой некоторые функции. К ним относятся функции:

– Сортировка и вывод переменных окружения;
– Чтение файла окружения;
– Создание дочернего процесса;
– Обработка команд пользователя.

Функция print_env_from_envp

Функция перебирает массив переменных окружения envp и выводит каждую из них на стандартный вывод (stdout). Это позволяет программе отобразить весь набор переменных окружения, переданных при запуске. Используется цикл, в котором указатель env последовательно проходит по всем строкам массива, а printf выводит их на экран.

Функция print_env_from_file

Функция открывает указанный файл и построчно читает из него имена переменных окружения. Затем для каждой переменной вызывает getenv, чтобы получить её значение. Если переменная существует, выводит её в формате ключ=значение. Если файл не удаётся открыть, функция выводит сообщение об ошибке через perror. После завершения чтения файл закрывается.

Функция spawn_child

Функция создаёт новый дочерний процесс с помощью `fork()`. Если создание процесса (`fork()`) не удалось, выводится сообщение об ошибке. В дочернем процессе программа формирует путь к исполняемому файлу (`child_exec`) и имя процесса (`child_name`). Затем выполняет `execve` или `execl` в зависимости от переданного параметра `mode`:

`+` — дочерний процесс получает переменные окружения из файла `ENV_FILE`.

`*` — передаёт дочернему процессу переменные окружения из массива `child_env`.

`&` — также использует `child_env`, но вариант может отличаться по логике в дальнейшем.

Если `exec` завершился с ошибкой, программа выводит сообщение и завершает процесс с `exit(EXIT_FAILURE)`. Для предотвращения бесконечного создания процессов предусмотрено ограничение в 99 дочерних процессов.

Функция load_env_from_file

Функция загружает переменные окружения из файла `ENV_FILE`. Открывает файл и построчно читает имена переменных, затем с помощью `getenv` получает их значения. Если переменная существует, формирует строку ключ=значение, выделяя память для неё, и добавляет в массив `new_env`. Если память не удалось выделить, выводится сообщение об ошибке. После обработки всех строк массив завершается `NULL` и возвращается вызывающей функции. Если файл не удаётся открыть, функция возвращает `NULL`.

Функция free_env

Функция освобождает память, выделенную под массив переменных окружения. Для этого она проходит по всем элементам массива и освобождает их с помощью `free()`, а затем освобождает сам массив. Это предотвращает утечки памяти.

Функция handle_keyboard

Функция обрабатывает ввод с клавиатуры, позволяя пользователю запускать дочерние процессы. Выводит инструкцию по управлению:

`+` — создаёт процесс с переменными окружения из файла;

`*` — создаёт процесс с переменными окружения из массива `child_env`;

`&` — также использует `child_env`, но возможно с другой логикой;

`q` — завершает работу.

Функция main (parent.c)

Функция main загружает переменные окружения из файла с помощью load_env_from_file(), и если загрузка не удалась, завершает программу с ошибкой. Далее проверяется наличие переменной окружения CHILD_PATH, определяющей путь к исполняемому файлу дочернего процесса; если она не задана, программа выводит ошибку и освобождает память перед завершением. Затем вызывается handle_keyboard(), обрабатывающая ввод пользователя и позволяющая создавать дочерние процессы с разными вариантами окружения. После выхода из режима управления клавиатурой while (wait(NULL)> 0) ожидает завершения всех дочерних процессов. Наконец, перед выходом программа выводит сообщение "Parent exiting.", освобождает память, выделенную под переменные окружения, и завершает выполнение с кодом EXIT_SUCCESS.

Функция main (child.c)

Функция main в этом коде выполняет логику дочернего процесса. Сначала она выводит информацию о себе, включая имя (argv[0]), идентификатор процесса (getpid()) и идентификатор родительского процесса (getppid()). Далее, если в аргументах командной строки передан файл (argc > 1), вызывается print_env_from_file(argv[1]), что означает, что программа загружает и выводит переменные окружения из указанного файла. В противном случае (argc == 1) используется print_env_from_envp(envp), которая выводит переменные окружения, переданные в envp. После выполнения этих операций программа завершает работу с кодом EXIT_SUCCESS.

2.3 Описание запуска и сборки

Для сборки проекта используется утилита cmake и файл CMakeLists.txt, которая автоматически создает makefile, с помощью которого производится компиляция исходного кода и создание исполняемых файлов из child.c и parent.c. Запускать cmake следует в директории build для структуризации проекта (если находитесь в папке build: $cmake ../CmakeLists.txt; далее: $make). Перед запуском программы parent в ее окружении пользователем создается переменная CHILD_PATH с именем каталога, где находится программа child (если находитесь в папке build: $export CHILD_PATH=./).

Для удаления скомпилированного файла и очистки директории build предусмотрена команда make clean. Это позволяет поддерживать рабочую среду в чистоте и гарантировать корректность последующих сборок.

3 РЕЗУЛЬТАТЫ РАБОТЫ ПРОГРАММЫ

```
=== Parent environment ===
CHILD_PATH=./
CHROME_DESKTOP=code.desktop
COLORTERM=truecolor
DBUS_SESSION_BUS_ADDRESS=unix:path=/run/user/1000/bus
DEBUGINFOD_URLS=https://debuginfod.ubuntu.com
DESKTOP_SESSION=ubuntu
DISPLAY=:1
FONTCONFIG_FILE=/etc/fonts/fonts.conf
FONTCONFIG_PATH=/etc/fonts
GDK_BACKEND=x11
```

Рисунок 3.1 – Запуск с показом окружения для parent

```
+
Child process: child_00 (PID: 4861, PPID: 4799)
SHELL=/usr/bin/zsh
HOME=/home/vadim
LOGNAME=vadim
LANG=en_US.UTF-8
TERM=xterm-256color
USER=vadim
LC_COLLATE=C
PATH=/home/vadim/.local/bin:/usr/local/sbin:/usr/local/bin:/usr/
sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin:/
snap/bin
```

Рисунок 3.2 – Результат при символе «+»

```
*
Child process: child_01 (PID: 4906, PPID: 4799)
SHELL=/usr/bin/zsh
HOME=/home/vadim
LOGNAME=vadim
LANG=en_US.UTF-8
TERM=xterm-256color
USER=vadim
LC_COLLATE=C
PATH=/home/vadim/.local/bin:/usr/local/sbin:/usr/local/bin:/usr/
sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin:/
snap/bin
```

Рисунок 3.3 – Результат при символе «*»

```
&
Child process: child_02 (PID: 4956, PPID: 4799)
SHELL=/usr/bin/zsh
HOME=/home/vadim
LOGNAME=vadim
LANG=en_US.UTF-8
TERM=xterm-256color
USER=vadim
LC_COLLATE=C
PATH=/home/vadim/.local/bin:/usr/local/sbin:/usr/local/bin:/usr/
sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin:/
snap/bin
```

Рисунок 3.4 – Результат при символе «&» и выход из программы

4 ВЫВОД

В ходе изучения системных вызовов было рассмотрено создание процессов с помощью `fork()`, замена текущего процесса на новый с использованием `execve()`, а также получение идентификаторов процессов через `getpid()` и `getppid()`. Также была изучена функция `getenv()`, позволяющая извлекать значения переменных окружения, и рассмотрена связь системного вызова `execve()` с функцией `main()` языка C, что позволяет запускать новые программы с заданными аргументами и окружением.