Семинар #8: Процессы. Практика.

Как сдавать задачи

Для сдачи ДЗ вам нужно создать репозиторий на GitLab (если он ещё не создан) под названием `devtools-homework`. Структура репозитория должна иметь вид:

```
├── seminar8_processes/
│   ├── 01.sh
│   ├── 02.sh
│   └── ...
└── ...
```

Для каждой задачи, если в самой задаче не сказано иное, нужно создать 1 скрипт с расширением `.sh` и шебангом в начале скрипта. Если задача делится на подзадачи нужно, если в самой задаче не сказано иное, создать скрипт для каждой подзадачи. Названия файлов решений для всех задач/подзадач должны начинаться с номера задачи, например `01.sh` или `04b.sh`, даже если в условии задачи используется другое имя для скрипта. Если в задаче встречается вопрос, то на этот вопрос нужно ответить в комментариях (начинаются с `#`) скрипта.
Если в задаче требуется в определённый момент перейти в другой терминал, отметьте это в файле решения, используя комментарий:

```
# Переходим во второй терминал
...
# Возвращаемся в первый терминал
```

В некоторых задачах используются готовые программы на языках bash и C. Найти эти программы можно в `seminar8_processes/practice/code`.

Job Control

Задача 1. Запуск в фоновом режиме

(a) Создадим скрипт `counter.sh` который будет печатать на экран возрастающие целые числа с шагом в две секунды.

```
#!/bin/bash
i=0
while true; do
    echo "$i"
    sleep 2
    ((i += 1))
done
```

Напишите или скачайте этот скрипт с репозитория и дайте ему права на исполнение.

(b) Запустите скрипт:

```
$ ./counter.sh
```

В результате на экран будут печататься числа. Bash будет занят выводом чисел на экран, поэтому в данной оболочке мы ничего сделать не сможем, пока не завершим программу. Завершите исполнение программы, используя `Ctrl-C`.

(c) Запустите скрипт в фоновом режиме:

```
$ ./counter.sh &
```

Bash запустит программу в фоновом режиме. В этом случае bash будет свободен и вы сможете выполнять другие команды. Правда, так как `counter.sh` всё ещё выводит числа на экран, то работа с оболочкой может быть затруднена. Завершить программу, используя `Ctrl-C` в данном случае невозможно. Чтобы завершить процесс, его нужно сначала перевести из фонового (background) режима в передний (foreground) режим, используя команду `fg` и после этого завершить с помощью `Ctrl-C`.

(d) Запустите скрипт с выводом в файл:

```
$ ./counter.sh > a.txt
```

В результате в файл `a.txt` будут записываться числа. Bash будет занят записью чисел в файл, поэтому в данной оболочке мы ничего сделать не сможем. Но можно проверить, что числа печатаются в файл, используя другой терминал. Откройте другой терминал, зайдите в ту же директорию и проверьте, что в файл `a.txt` в данный момент производится запись, несколько раз вызвав `cat`. Также можно использовать команду:

```
$ watch -n 2 cat a.txt
```

которая будет автоматически повторять команду `cat a.txt` каждые 2 секунды. Но, так как запись происходит в конец файла, то скорей всего лучше использовать команду `tail` вместо `cat`.

```
$ watch -n 2 tail a.txt
```

Задача по выводу последних строк изменяемого файла используется настолько часто, что команде `tail` дали специальную опцию для этого случая:

```
$ tail -f a.txt
```

Которая выводит последние строки файла, после изменения этого файла.

Закройте второй терминал, вернитесь на первый и завершите процесс `counter.sh`.

(e) Запустите скрипт с выводом в файл в фоновом режиме:

```
$ ./counter.sh > a.txt &
```

В результате в файл `a.txt` будут записываться числа. В этом случае bash будет свободен и вы сможете выполнять другие команды. Проверьте, что программа `counter` работает и в файл записываются числа, используя команду `tail -f`. Чтобы завершить процесс, его нужно перевести из фонового (background) режима в передний (foreground) режим, используя команду `fg` и после этого завершить с помощью `Ctrl-C`.

(f) Запустите скрипт с выводом в файл в фоновом режиме:

```
$ ./counter.sh > a.txt &
```

Выполните команду `ps`, которая покажет все программы, запущенные в данном терминале. Скорей всего на экран выведется информация о процессах `bash` (оболочка), `counter.sh`, `sleep` (запушенный внутри `counter.sh`) и только что запущенный `ps`. Для каждого процесса, будет выведен его идентификатор PID (*process identifier*). Используйте полученный PID процесса `counter.sh` и команду `kill`, чтобы завершить процесс. Команда `kill <PID>` пошлёт сигнал `SIGTERM` процессу с идентификатором `<PID>`, что завершит этот процесс. Проверьте, что процесс завершился и запись в `a.txt` больше не производится. Ещё раз выполните команду `ps` и убедитесь, что процесс `counter.sh` отсутствует в выводе.

Задача 2. Приостановка процесса

(a) **Простая приостановка**

- Запустите скрипт `counter.sh`:

  ```
  $ ./counter.sh
  ```

- Нажмите комбинацию клавиш `Ctrl-Z`, чтобы приостановить процесс. Теперь процесс приостановился и ничего не выводит на экран.
- Для возобновления работы процесса на переднем плане используйте команду `fg` (*foreground process*).

– Завершите процесс, используя `Ctrl-C`.

(b) **Передний и фоновый режимы**

– Запустите скрипт `counter.sh` с записью в файл:

```
$ ./counter.sh > a.txt
```

– Откройте новый терминал и запустите на нём просмотр конца файла `a.txt`, используя `tail -F a.txt`. Убедитесь, что в файл происходит запись. Не закрывайте этот терминал, на нём мы будем проверять, что наш процесс работает.

– Перейдите на основной терминал. Нажмите комбинацию клавиш `Ctrl-Z`, чтобы приостановить процесс. Теперь процесс приостановился и ничего не будет записывать в файл. Убедитесь в этом на втором терминале.

– Используйте команду `fg`, чтобы возобновить работу процесса на переднем плане. Убедитесь, что процесс начал работу, проверив вывод команды `tail` на втором терминале.

– Снова приостановите процесс, используя `Ctrl-Z`. Используйте команду `bg`, чтобы возобновить процесс, но теперь в фоновом режиме. Убедитесь, что процесс начал работать, посмотрев вывод на другом терминале.

– Переведите процесс на передний план и завершите его.

(c) **Список задач**

– Завершите все предыдущие процессы и запустите скрипт три раза в фоновом режиме:

```
$ ./counter.sh > a.txt &
$ ./counter.sh > b.txt &
$ ./counter.sh > c.txt &
```

– Убедитесь, что все задачи работают в фоновом режиме, проверив запись в соответствующие файлы.

– Выполните команду `jobs`, чтобы посмотреть все фоновые или приостановленные задачи в данной оболочке. Символом `+` будет отображаться последняя добавленная в `jobs` задача, а символом `-` предпоследняя.

– Переведите задачу записи в `a.txt` на передний план, используя команду:

```
$ fg %N   # вместо N нужно подставить номер задачи в jobs
```

Остановите эту задачу с помощью `Ctrl-Z` и снова посмотрите на список задач.

– Теперь задача записи в `a.txt` приостановлена. Обратите внимание, что рядом с задачей записи в `a.txt` стоит символ `+`, так как эта последняя добавленная задача. Возобновите работу этой программы в фоновом режиме, используя:

```
$ bg %N   # вместо N нужно подставить номер задачи в jobs
```

или, так как эта задача является последней добавленной, то можно использовать `bg` без указания номера. В этом случае операция будет применена к последней добавленной задаче. Снова посмотрите на все задачи, используя `jobs`.

– Используйте команду

```
$ kill %N   # вместо N нужно подставить номер задачи в jobs
```

чтобы завершить задачу записи в файл `c.txt`. Снова посмотрите на все задачи.

– Используйте команду

```
$ kill -STOP %N   # вместо N нужно подставить номер задачи в jobs
```

чтобы остановить задачу записи в файл `b.txt`. Посмотрите на все задачи. Используйте `bg` или

```
$ kill -CONT %N   # вместо N нужно подставить номер задачи в jobs
```

чтобы возобновить приостановленную задачу.

Просмотр процессов

Задача 3. Список процессов

(a) Запустите скрипт `counter.sh`. Он должен работать во время решения этой задачи.

(b) Перейдите в другой терминал и напечатайте все процессы данного терминала, используя `ps`. При этом процесс `counter.sh` не будет отображаться, так как он запущен на другом терминале.

(c) Напечатайте все процессы, используя `ps -ef`. Эта команда покажет информацию о uid, pid, ppid, c, stime, tty, time, cmd всех процессов в системе. Найдите среди этих процессов процесс `counter.sh`. Чему равны `PID` и `PPID` для этого процесса?

(d) Напечатайте все процессы, используя `ps aux`. Эта команда покажет информацию о user, pid, pcpu, pmem, vsz, rss, tty, stat, start, time, cmd всех процессов в системе. Найдите среди этих процессов процесс `counter.sh`. В каком состоянии находится процесс `counter.sh` (поле STAT)?

(e) Используйте команду `pstree`, чтобы напечатать информацию о всех процессах в древовидном виде. Найдите среди этих процессов процесс `counter.sh`. Какие процессы являются предками процесса `counter.sh`? Программа `pstree` может быть не установлена в системе. Если это так, то нужно будет её установить.

(f) Напечатайте все процессы, показав информацию только о pid, ppid, pcpu, pmem, cmd.

(g) Напечатайте все процессы, показав только pid, ppid, pcpu, pmem, cmd и отсортировав по полю pmem.

(h) Используйте `ps` с опцией `-p`, чтобы напечатать информацию uid, pid, ppid, pcpu, pmem, cmd только для процесса `counter.sh`.

(i) Используйте `ps` вместе с командой `grep`, чтобы напечатать информацию uid, pid, ppid, pcpu, pmem, cmd только для `counter.sh`.

(j) Используйте `ps`, чтобы напечатать только одно число – PPID процесса `counter.sh`. Нужно напечатать только одно число, без какого-либо текста, в том числе без заголовка.

(k) Используйте команду `kill`, чтобы остановить процесс `counter.sh` на другом терминале. На другом терминале убедитесь, что процесс остановлен и находится в списке jobs. Ещё раз используйте `kill`, чтобы возобновить работу скрипта `counter.sh`.

(l) Используйте команду `kill`, чтобы завершить процесс `counter.sh`.

(m) Запустите скрипт `counter.sh` заново на другом терминале. Используйте команду `pkill`, чтобы завершить процесс `counter.sh`

Задача 4. ID родителя, ID процесса и ID ребёнка

Напишите скрипт, которых будет делать следующее:

1. Создавать ребёнка, запуская скрипт `counter.sh` в фоновом режиме. Вывод ребёнка должен быть перенаправлен в файл `a.txt`.
2. Печатать на экран `PID` родителя данного процесса, а также команду с помощью которой был запущен родитель (используйте `ps -o cmd`).
3. Печатать на экран `PID` самого процесса, а также команду с помощью которой он был запущен.
4. Печатать на экран `PID` ребёнка данного процесса, а также команду с помощью которой он был запущен.

Задача 5. Родословная

Напишите скрипт, который будет принимать через аргумент PID процесса и выводить рекурсивно PID его родителей и команды, с помощью которых были все процессы запущены. Последний родитель будет иметь имя `systemd` (или `/sbin/init`) и иметь PID равный 1.

Группы процессов и сессии

Задача 6. Группа процессов

Когда оболочка запускает команду, она создаёт новую группу процессов для этой команды. Все дочерние процессы, порождённые этой командой, попадают в одну группу. Создайте скрипт `create_group.sh`, использующий скрипт `counter.sh` из предыдущих заданий (он должен находиться в той же директории) и содержащий:

```
#!/bin/bash
./counter.sh &
./counter.sh &
./counter.sh
```

Этот скрипт запускает два фоновых процесса `counter.sh` и один передний процесс `counter.sh`. Все четыре процесса (сам скрипт и три ребёнка) будут находиться в одной группе процессов.

(a) Запустите скрипт `create_group.sh`.

(b) Откройте другой терминал и используйте

```
$ ps -e -o pid,ppid,pgid,comm
```

чтобы распечатать информацию о `PID`, `PPID` и `PGID` (*process group id*) процессов, а также имя команды, запустившей процесс. Какие процессы были созданы после запуска скрипта `create_group.sh`? Убедитесь, что все созданные процессы принадлежат одной группе процессов. Какой процесс является лидером этой группы процессов?

(c) Завершите процесс скрипта `create_group.sh` по его `PID`, послав ему сигнал с помощью команды `kill`. Завершились ли при этом дочерние процессы скрипта? Если дочерние процессы не завершились, то какой процесс является их родителем? Если дочерние процессы не завершились, то завершите их с помощью `pkill`.

(d) Заново запустите скрипт `create_group.sh`. Используйте `ps`, чтобы узнать `PGID` новой группы процессов. Завершите сразу все процессы одной группы, послав им всем сигналы. Используйте только один вызов команды `kill`.

(e) Заново запустите скрипт `create_group.sh`. Если сейчас нажать `Ctrl-C`, то какие сигналы и каким процессам будут посланы? Какие процессы при этом завершатся?

Задача 7. Сессия

Все процессы, которые были запущены из одного терминала по умолчанию будут принадлежать одной сессии. Одна сессия может содержать несколько групп процессов.

(a) Используйте скрипт `create_group.sh`, чтобы создать две группы процессов в текущей сессии:

```
$ /create_group.sh > a.txt &
$ /create_group.sh > b.txt &
```

(b) Используйте

```
ps -e -o pid,ppid,pgid,sid,comm
```

чтобы распечатать информацию о `PID`, `PPID`, `PGID` и `SID` (*session id*) процесса, а также имя команды, запустившей процесс. Убедитесь, что значения `PGID` одинаково для процессов, находящихся в одной группе. Убедитесь, что значения `SID` одинаковы для всех созданных процессов. Кто является лидером сессии?

(c) Если сейчас закрыть терминал, то какие сигналы и каким процессам будут посланы? Какие процессы при этом завершатся?

Сигналы

Задача 8. Перехватчик сигналов

Напишите скрипт `signal_catcher.sh` на основе `counter.sh`. Скрипт `signal_catcher.sh` должен выводить числа на экран и одновременно перехватывать сигналы. При получении сигнала скрипт не завершается, а выводит соответствующее сообщение. Например, при сигнале `SIGTERM` он должен напечатать:

```
I caught SIGTERM
```

Скрипт должен ловить сигналы: `SIGINT`, `SIGTERM`, `SIGHUP`, `SIGTSTP`, `SIGCONT`. Используйте команду `trap`.

Задача 9. Работаем с перехватчиком сигналов

(a) Запустите скрипт `signal_catcher.sh`. Откройте другой терминал и попытайтесь завершить процесс с помощью команды:

```
$ kill <PID>
```

Что напечатает скрипт в этом случае?

(b) Отправьте процессу скрипта `signal_catcher.sh` сигналы `SIGINT` и `SIGTSTP`, используя команду `kill`. Что напечатает скрипт в этом случае?

(c) Завершите процесс `signal_catcher.sh` с помощью сигнала `SIGKILL`.

(d) Заново запустите скрипт `signal_catcher.sh`. После запуска скрипта нажмите `Ctrl-C`. Что напечатает программа и почему? Если быстро нажать `Ctrl-C` несколько раз подряд, то программа будет печатать числа быстрее чем раз в две секунды. Почему?

(e) Нажмите `Ctrl-Z` в терминале скрипта. Скрипт остановится, хотя он должен ловить сигнал `SIGTSTP`. Почему скрипт остановился? Восстановите работу скрипта, не завершая его и не запуская его заново.

(f) Запустите скрипт следующим образом:

```
$ signal_catcher.sh > sc.txt
```

После этого закройте терминал. Завершится ли процесс `signal_catcher.sh` после закрытия терминала? Проверьте жив ли процесс, посмотрев происходит ли печать в файл `sc.txt`.

Задача 10. Перехват или игнорирование

(a) **Перехват сигнала**

Пусть есть скрипт `intercept.sh` который перехватывает сигнал `SIGINT`:

```
#!/bin/bash
trap 'echo I caught SIGINT' INT
sleep 1000
```

Если запустить этот скрипт и нажать `Ctrl-C`, то завершится ли скрипт? Объясните поведение скрипта в этом случае.

(b) **Блокирование сигнала**

Пусть есть скрипт `ignore.sh` который блокирует сигнал `SIGINT`:

```
#!/bin/bash
trap '' INT
sleep 1000
```

Если запустить этот скрипт и нажать `Ctrl-C`, то завершится ли скрипт? Объясните поведение скрипта в этом случае.

Задача 11. Отсоединённый процесс

Если запустить скрипт `counter.sh` вот так:

```
$ ./counter.sh > c.txt
```

то после закрытия терминала процессу `counter.sh` будет послан сигнал `SIGHUP` и он завершиться, так как этот процесс не перехватывает и не блокирует сигналы. Используя команду `nohup`, запустите скрипт `counter.sh` с блокировкой сигнала `SIGHUP`. После этого закройте терминал, откройте новый терминал и убедитесь, что процесс `counter.sh` всё ещё жив. Завершите этот процесс с помощью `kill`.

Задача 12. Сигналы от дочерних процессов

Пусть есть скрипт `random_sleeper.sh`, который просто спит случайное число секунд от 1 до 10:

```
#!/bin/bash
sleep $((RANDOM % 10 + 1))
```

Напишите скрипт `waiter.sh`, который будет принимать число `n` через аргументы командной строки и запускать скрипт `random_sleeper.sh` параллельно `n` раз. Скрипт `waiter.sh` должен дожидаться всех своих дочерних процессов – используйте команду `wait`. Также скрипт должен отслеживать завершения всех дочерних процессов с помощью `trap`. После того как любой из дочерних процессов завершается, `waiter.sh` должен сразу печатать на экран строку вида:

```
Child k of n is finished.
```

Пример запуска такого скрипта:

```
$ ./waiter.sh 3
Child 1 of 3 is finished.
Child 2 of 3 is finished.
Child 3 of 3 is finished.
```

Состояния процессов

Задача 13. Спящий и остановленный процесс

Напишите скрипт, который создаёт спящий процесс и остановленный процесс. После этого скрипт должен напечатать `PID` этих процессов. Два созданных процесса должны существовать после того, как скрипт завершился. Используйте `ps aux`, чтобы просмотреть состояния процессов.

Задача 14. Состояния процессов

Объясните, что означают следующие состояния процессов в выводе команды `ps`:

(a) `R`

(b) `S`

(c) `T`

(d) `D`

(e) `Z`

(f) `Rl`

(g) `Ss`

(h) `SNsl`

(i) `I` (для потока ядра)

(j) `I<` (для потока ядра)

Задача 15. Зомби процесс

Обычно родитель ждёт завершения дочернего процесса, чтобы получить код возврата и понять, успешно ли он завершился. От этого может зависеть дальнейшее поведение родителя. Но если ребёнок завершился, а родитель не вызвал команду или системный вызов wait для чтения кода возврата, процесс остаётся в особом состоянии — зомби (Z), пока родитель не прочитает его результат или сам не завершится.

Данная программа на языке C создаёт зомби-процесс (создать зомби-процесс в bash может быть непросто или невозможно в некоторых системах):

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main()
{
    pid_t pid = fork();
    if (pid > 0)
    {
        sleep(1000);
    }
    else if (pid == 0)
    {
        exit(0);
    }
}
```

Системный вызов `fork` создаёт ребёнка процесса, путём клонирования текущего процесса. Можно считать, что в `fork` заходит один процесс (родительский), а из `fork` выходят два процесса (родительский и дочерний). В дочернем процессе `fork` возвращает `0`. В родительском – значение `PID` ребёнка. В данной программе после `fork` родительский процесс засыпает на 1000 секунд, а дочерний процесс сразу пытается завершиться с помощью `exit`. Обычно, родительский процесс должен дождаться прочитать код возврата дочернего процесса, но он этого не делает, в результате дочерний процесс становится зомби процессом.

(a) Скомпилируйте программу, используя компилятор `gcc`.

(b) Запустите программу и убедитесь, что дочерний процесс стал зомби процессом.
Запомните PID родительского процесса и дочернего зомби процесса.

(c) Можно ли завершить зомби процесс, послав ему сигнал `SIGTERM`?

(d) Можно ли завершить зомби процесс, послав ему сигнал `SIGKILL`?

(e) Как можно завершить зомби процесс?

Мониторинг

Задача 16. Программа `top`

Программа `top` используется для мониторинга ресурсов процессов. Объясните, что означают следующие поля в выводе `top`.

(a) `PID`

(b) `USER`

(c) `PR`

(d) `NI`

(e) `VIRT`

(f) `RES`

(g) `SHR`

(h) `S`

(i) `%CPU`

(j) `%MEM`

(k) `TIME+`

(l) `COMMAND`

Задача 17. Директория `/proc`

- Запустите программу `counter.sh` и найдите `PID` процесса.
- Перейдите в директорию виртуальной файловой системы `proc`, соответствующей этому процессу.

  `/proc/<PID>/`

- Что содержится в следующих файлах этой директории?
  - `cmdline`
  - `environ`
  - `maps`
- Куда указывают следующие мягкие ссылки в этой директории?
  - `cwd`
  - `exe`
  - `fd/1`
- Что означают следующие поля в файле `status`?
  - `Name`
  - `State`
  - `Uid / Gid`
  - `VmSize`
  - `VmRSS`
  - `VmData, VmExe, VmStk`
  - `FDSize`