Задание 1

1. Функция kill

Когда вы выполняете команду «kill», вы на самом деле посылает е сигнал к системе и даете указание расторгнуть работу некорректной роботы приложения. Существует в общей сложности 60 сигналов, которые можно использовать

1. Неблокирующий wait c WNOHANG

Это такой wait, который при отсутствии потомков, завершивших выполнений, немедленно возвращается

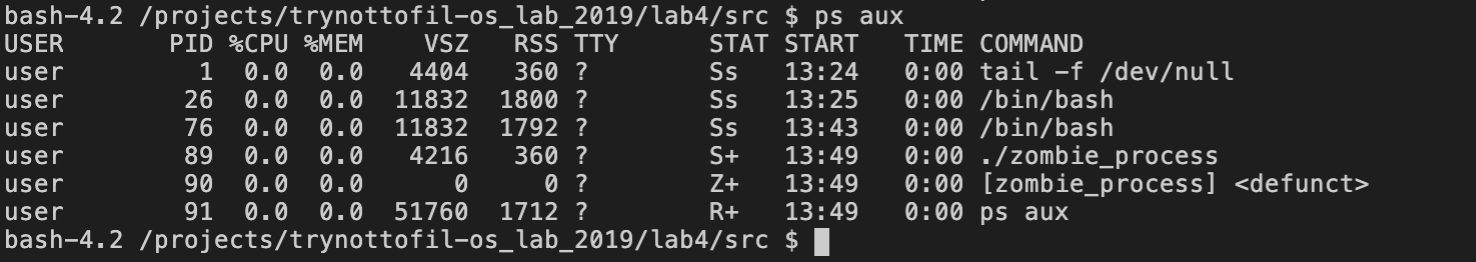
1. Функция alarm, сигнал SIGALRM, функция signal

Функция alarm () указывает системе сгенерировать сигнал SIGALRM через время, заданное аргументом seconds (в секундах). Из-за накладных расходов системы сигнал может быть сгенерирован несколько позже.

Задание 2

1. Что такое зомби процессы, как появляются, как исчезают.

Обычный процесс превращается в зомби, когда дочерний процесс получает команду на завершение, он освобождает все используемые ресурсы и продолжает как зомби существовать в системе с ранее присвоенным ему PID. Далее с помощью сигнала SIGCHLD система уведомляет родительский процесс о завершении зомби процесса. Если по какой-либо причине родительский процесс игнорирует этот сигнал, то зомби процесс так и продолжает отображаться в системе.



Задание 3

1. Работа виртуальной памяти.

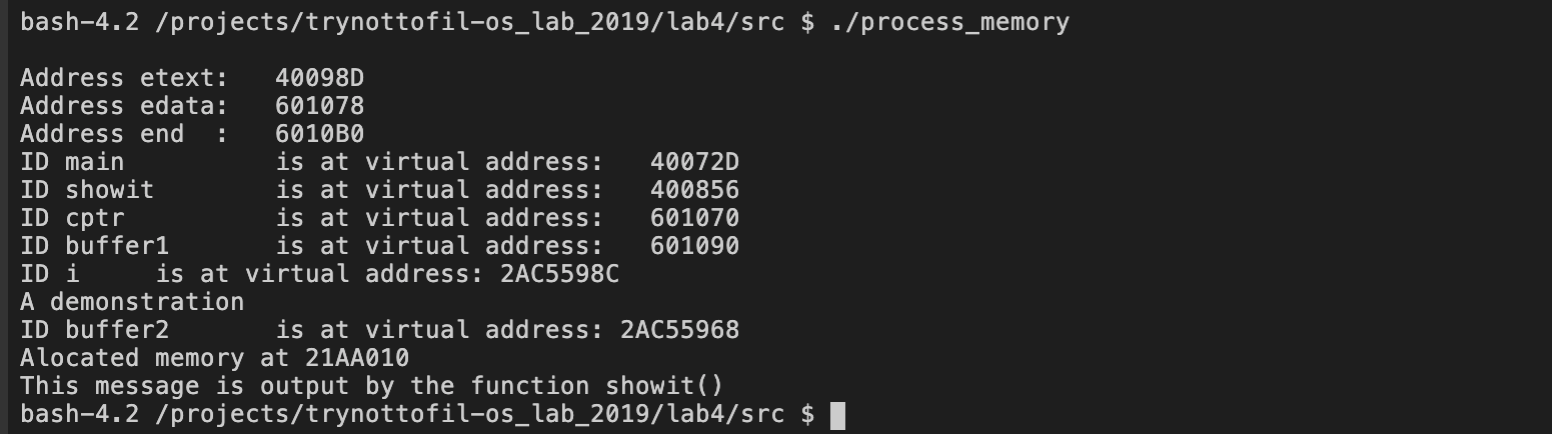
Linux поддерживает *виртуальную память*, то есть использует диск как расширение оперативной памяти так, чтобы эффективный размер пригодной для использования памяти был больше размера памяти, реально установленной в системе. Ядро пишет содержимое блока памяти, неиспользуемого в настоящее время на жесткий диск так, чтобы память могла использоваться для другой цели. Когда первоначальное содержание понадобится снова, оно будет считано обратно в память. Этот механизм сделано полностью прозрачным для пользователя; программы, запущенные под Linux, видят только большой доступный объем памяти, и не обращают внимания на то, что их части время от времени оказываются на диске. Конечно, чтение и запись жесткого диска медленнее, чем использование реальной памяти, так что программы не выполняются как быстро. Часть жесткого диска, который используется как виртуальная память, называется *свопом*.

Linux может использовать нормальный файл в любой файловой системе или отдельном разделе для свопа. Раздел для свопа быстрее, но проще изменить размер файла подкачки (нет никакой потребности к перераспределению целого жесткого диска и, возможно, устанавке всего с нуля). Когда Вы знаете, сколько места для свопа Вам надо, Вы должны использовать раздел для свопа, но если Вы не уверенные, лучше сначала использовать файл подкачки сначала, поработать с системой некоторое время, чтобы Вы могли получить представление о том, сколько места Вам нужно, и затем, когда вы будете уверены относительно размера, создать раздел для свопа.

Вы должны также знать, что Linux позволяет использовать несколько разделов и/или файлов подкачки в одно и то же время. Это означает, что, если Вы только иногда нуждаетесь в большом количестве виртуальной памяти, Вы можете устанавливать дополнительный файл подкачки в таких случаях, вместо того, чтобы хранить такой файл постоянно.

Примечание относительно терминологии операционной системы: информатика обычно различает между подкачкой (запись целого процесса, чтобы менять свободное место) и листание (запись только частей фиксированного размера, обычно несколько килобайт, одновременно). Листание обычно более эффективно, и именно его Linux и использует, но традиционно терминология в Linux употребляет все же термин подкачки.





Адрес etext - адрес сразу за концом сегмента команд.

Адрес edata - адрес сразу за концом области инициализированных данных.

Адрес end - адрес сразу за концом области неинициализированных данных.

Функция main() расположена в (виртуальном) сегменте команд, т.к адрес main()<адреса сразу за концом сегмента команд.

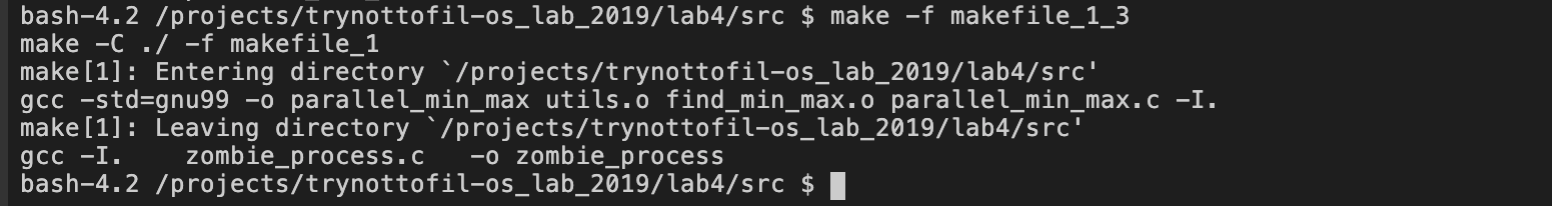
Функция showit()рпасположена в сегменте команд

Указатель на глобальную “строку” расположен в сегменте памяти инициализированных данных

Массив символов buffer1 лежит в сегменте неинициализированных данных

Далее переменная i лежит где-то в стеке. А динамический массив buffer2 соответственно где-то в куче.

Задание 4:



Задание 5 :

1. POSIX threads: как создавать, как дожидаться завершения.

В начале создается потоковая функция. Затем новый поток создается функцией pthread\_create(), объявленной в заголовочном файле [pthread.h](http://pubs.opengroup.org/onlinepubs/9699919799/basedefs/pthread.h.html). Далее, вызывающая сторона продолжает выполнять какие-то свои действия параллельно потоковой функции.

#include <pthread.h>

int pthread\_create(pthread\_t \*thread, const pthread\_attr\_t \*attr, void \*(\*start)(void \*), void \*arg);

Поток завершает выполнение задачи когда:

* потоковая функция выполняет return и возвращает результат произведенных вычислений;
* в результате вызова завершения исполнения потока pthread\_exit();
* в результате вызова отмены потока pthread\_cancel();
* одна из нитей совершает вызов exit()
* основная нить в функции main() выполняет return, и в таком случае все нити процесса резко сворачиваются.

#include <pthread.h>

void pthread\_exit(void \*retval);

Функция pthread\_join() ожидает завершения потока обозначенного THREAD\_ID. Если этот поток к тому времени был уже завершен, то функция немедленно возвращает значение. Смысл функции в том, чтобы синхронизировать потоки.

#include <pthread.h>

int pthread\_join (pthread\_t THREAD\_ID, void \*\* DATA);

1. Как линковаться на бибилотеку pthread

Чтобы подключить библиотеку Pthread к программе, нужно передать компоновщику опцию -lpthread.  
gcc -o progtest -std=c99 -lpthread progtest.c

Задание 6:



