**Лабораторная работа №4**

**Задание 1**

**Необходимые знания**

*1.Функция kill.*

Когда вы выполняете команду «kill», вы на самом деле посылаете сигнал к системе и даете указание расторгнуть работу некорректной роботы приложения. Существует в общей сложности 60 сигналов, которые можно использовать, но самые главные – это SIGTERM и SIGKILL.

SIGTERM – Этот сигнал запрашивает остановку процесса который работает. Этот сигнал может быть проигнорирован. Процессу дается время, чтобы хорошо выключился. Когда программа хорошо выключается, это означает, что ей дано время, чтобы спасти его прогресс и освободить ресурсы. Другими словами, он не «forced» прекращение работы процесса.

SIGKILL – сигнал SIGKILL заставляет процесс прекратить выполнение своей работы немедленно. Программа не может игнорировать этот сигнал. Несохраненный прогресс будет потерян.

Функцию kill() можно использовать для принудительного завершения другого процесса. Эта функция отправляет сигнал процессам, заданным параметром pid. Параметр sig — это сигнал, предназначенный для отправки заданному процессу. Для уничтожения процесса параметр sig должен иметь значение SIGKILL. Чтобы иметь право отсылать сигнал процессу, вызывающий процесс должен обладать соответствующими привилегиями, либо его реальный или идентификатор эффективного пользователя должен совпадать с реальным или сохраненным пользовательским идентификатором процесса, который принимает этот сигнал. Вызывающий процесс может иметь разрешение на отправку процессам только определенных (а не любых) сигналов. При успешной отправке сигнала функция возвращает вызывающему процессу значение 0, в противном случае — число -1.

*2.Неблокирующий wait c WNOHANG.*

Функция wait приостанавливает выполнение текущего процесса до тех пор, пока дочерний процесс не завершится, или до появления сигнала, который либо завершает текущий процесс, либо требует вызвать функцию-обработчик. Если дочерний процесс к моменту вызова функции уже завершился (так называемый "зомби" ("zombie")), то функция немедленно возвращается. Системные ресурсы, связанные с дочерним процессом, освобождаются.

WNOHANG означает немедленное возвращение управления, если ни один дочерний процесс не завершил выполнение.

*3.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.

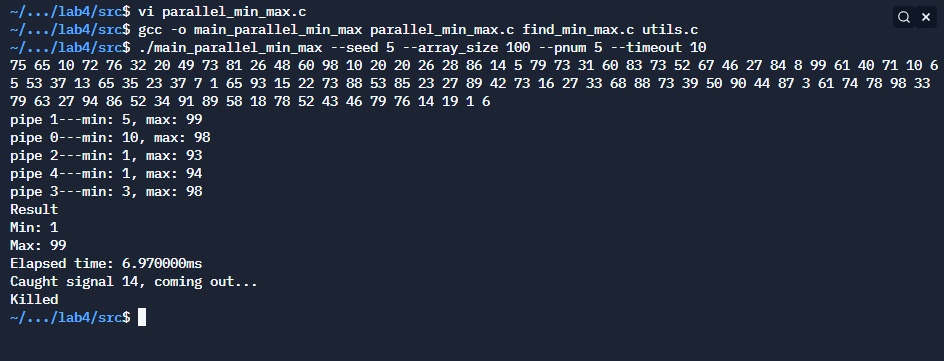
Повторный вызов функции alarm () приводит к отмене действия предыдущего вызова (если сигнал SIGALRM еще не был сгенерирован), и установке нового времени генерации сигнала.

Если аргумент seconds равен нулю, то функции alarm() только отменяет предыдущий вызов этой функции.

***Дополнить программу parallel\_min\_max.c из лабораторной работы №3, так чтобы после заданного таймаута родительский процесс посылал дочерним сигнал SIGKILL. Таймаут должен быть задан, как именной необязательный параметр командной строки (--timeout 10). Если таймаут не задан, то выполнение программы не должно меняться.***

***(Code in github)***

<https://github.com/lehoanggia2k/os_lab_2019>

****

**Задание 2**

**Необходимые знания**

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

- Зомби - это МЕРТВЫЕ процессы. Их нельзя «убить» (вы не можете убить МЕРТВУ). Все процессы в конечном итоге умирают, и когда они это делают, они становятся зомби. Они почти не потребляют ресурсов, чего и следовало ожидать, потому что они мертвы! Причиной для зомби является то, что родитель (процесс) зомби может получить информацию о состоянии выхода из зомби и статистику использования ресурсов. Родитель сообщает операционной системе, что зомби больше не нужен, используя один из системных вызовов wait ().

- Они появляются только тогда, когда их родительский процесс (то есть процесс, который fork()'ed они) жив, но еще не вызвал wait() системную функцию. После того, как родитель умирает, зомби wait()'ed за помощью init и исчезают.

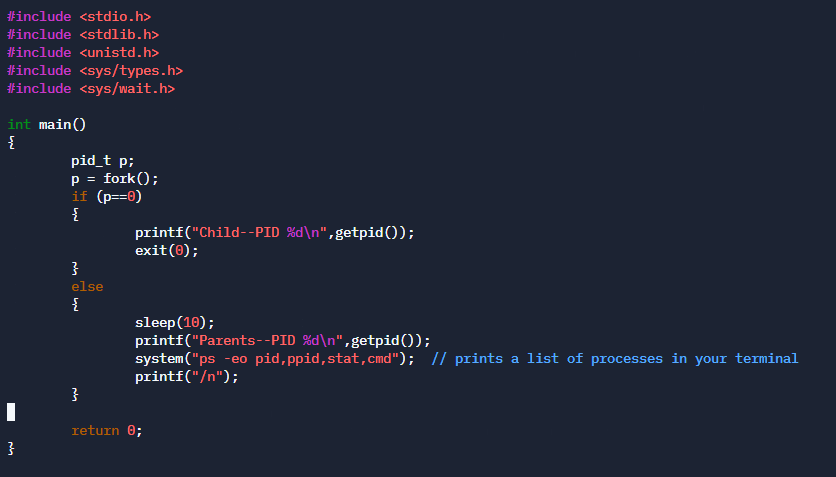
- Зомби процессы не используют системные ресурсы, поэтому их можно просто игнорировать.

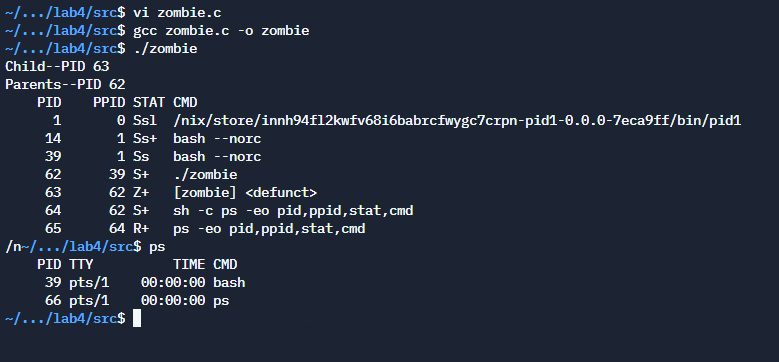
- Убить зомби процесс на прямую командой kill не получиться.

- В данном случае единственный вариант полностью убрать зомби процесс — убить или перезапустить его родительский процесс.

- Что бы убить зомби процесс, для начала нужно получить его PID

***Создать программу, с помощью которой можно продемонстрировать зомби процессы. Необходимо объяснить, как появляются зомби процессы, чем они опасны, и как можно от них избавиться.***

******

****

*Если дочерний процесс завершится до того, как родительский процесс вызовет wait (). Ядро решает эту проблему, возвращая дочерний процесс в состояние зомби.*

*Если процесс создает дочерний процесс без вызова wait (), когда дочерний процесс завершается, информация о зомби будет навсегда сохранена в таблице процессов ядра. Когда существует большое количество зомби, помимо таблицы процессов ядра, новые процессы не будут созданы.*

*У нас уже есть 2 способа избежать зомби: использовать функцию wait или waitpid и завершить родительский процесс (или дождаться завершения родительского процесса).*

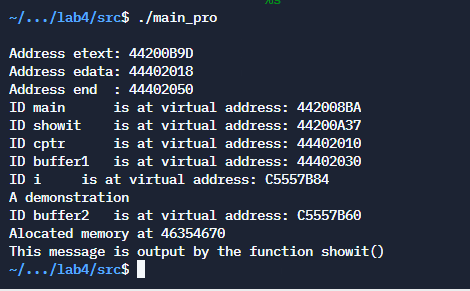
**Задание 3**

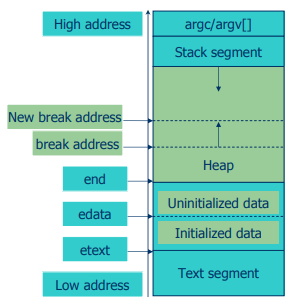
**Необходимые знания**

1.Работа виртуальной памяти.

-Виртуальная память — метод управления памятью, которая реализуется с использованием аппаратного и программного обеспечения компьютера. Она отображает используемые программами виртуальные адреса в физические адреса в памяти компьютера. Основная память представляется в виде непрерывного адресного пространства или набора смежных непрерывных сегментов. Операционная система осуществляет управление виртуальными адресными пространствами и соотнесением оперативной памяти с виртуальной. Программное обеспечение в операционной системе может расширить эти возможности, чтобы обеспечить виртуальное адресное пространство, которое может превысить объем оперативной памяти и таким образом иметь больше памяти, чем есть в компьютере. Виртуальная память позволяет модифицировать ресурсы памяти, сделать объём оперативной памяти намного больше, для того чтобы пользователь, поместив туда как можно больше программ, реально сэкономил время и повысил эффективность своего труда. “Открытие” виртуальной памяти внесло огромную контрибуцию в развитие современных технологий, облегчило работу как профессионального программиста, так и обычного пользователя, обеспечивая процесс более эффективного решения задач на ЭВМ.

***Скомпилировать process\_memory.c. Объяснить, за что отвечают переменные etext, edata, end.***

******

******

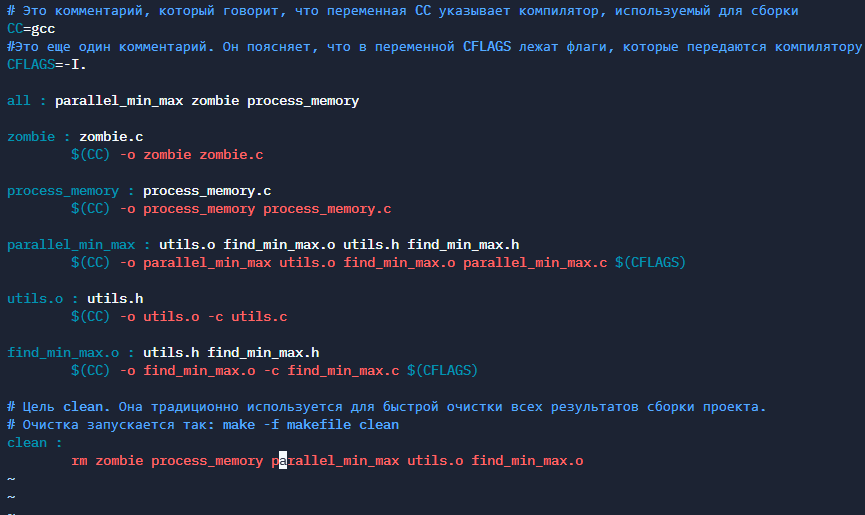
*etext*: this is the first address past the end of the text segment (the program code).

*edata*: this is the first address past the end of the initialized data segment.

*end:* this is the first address past the end of the uninitialized data segment (also known as the BSS segment).

**Задание 4**

***Создать makefile, который собирает программы из задания 1 и 3.***



**Задание 5**

**Необходимые знания**

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

-POSIX определяет набор интерфейсов (функций заголовочных файлов) для программирования потоков. Эти рекомендации носят название POSIX threads или Pthreads.

-Создается когда функция main() (in C) или PROGRAM (in fortran) вызываеьтся загрузчиком процесса. Функция main() может создавать дочерние threads. Если основной поток завершает работу, процесс прерывается даже если внутри процесса существуют другие потоки, если только не предприняты специальные действия. Для избегания прерывания процесса можно использовать pthread\_exit().

-Завершение потоков может происходить по двум причинам, а именно: завершилась функция, которую поток исполняет или (что аналогично) поток вызвал pthred\_exit(), поток прервали.

2.Как линковаться на библиотеку pthread.

-Библиотека Pthread может быть реализована на любом языке, но для соответствия стандарту POSIX она должна быть согласована со стандартизированным интерфейсом. Библиотека Pthread — не единственная реализация потокового API-интерфейса. Существуют другие реализации, созданные сторонними фирмами-производителями аппаратных и программных средств. Например, среда Sun поддерживает библиотеку Pthread и собственный вариант библиотеки потоков Solaris. В этой главе мы рассмотрим некоторые функции библиотеки Pthread, которые реализуют управление потоками.

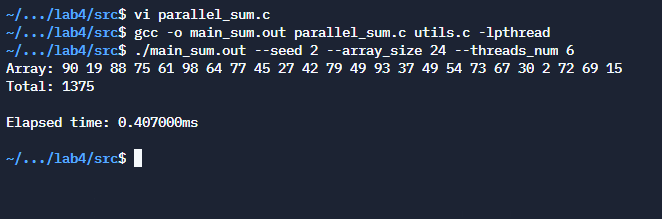
-Библиотека pthread предлагает множество функций, которые разработчик может использовать для того, чтобы создавать, уничтожать, присоединяться и синхронизировать потоки с помощью семафоров(мутексов).

***Доработать parallel\_sum.c так, чтобы:***

* ***Сумма массива высчитывалась параллельно.***
* ***Массив генерировался с помощью функции GenerateArray из лабораторной работы №3.***
* ***Программа должна принимать входные аргументы: количество потоков, seed для генерирования массива, размер массива (./psum --threads\_num "num" --seed "num" --array\_size "num").***
* ***Вместе с ответом программа должна выводить время подсчета суммы (генерация массива не должна попадать в замер времени).***
* ***Вынести функцию, которая считает сумму в отдельную библиотеку***

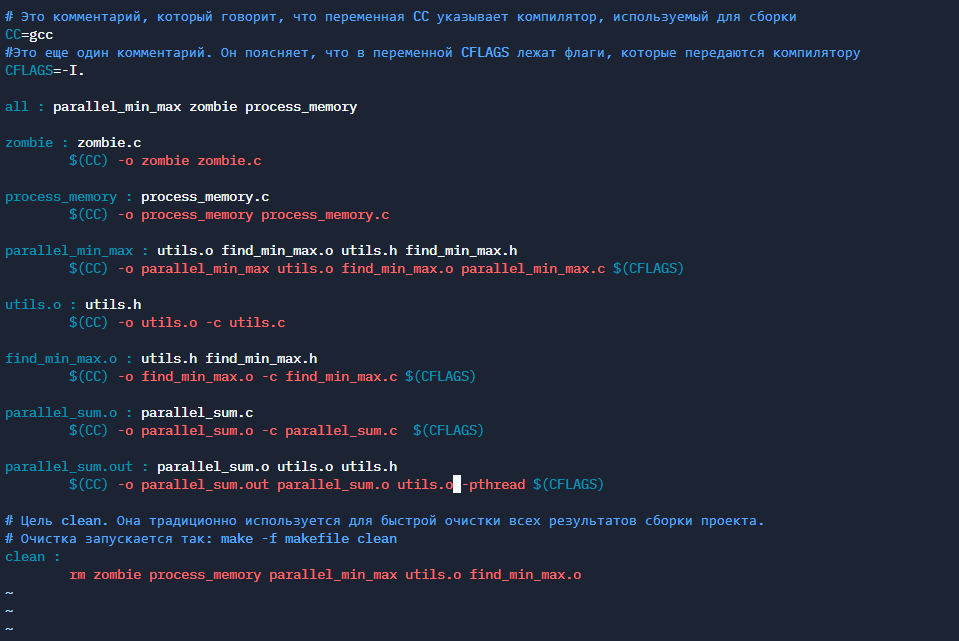
***Code in github***

***https://github.com/lehoanggia2k/os\_lab\_2019***



**Задание 5**

***Создать makefile для parallel\_sum.c.***

******

