**Отчет по лабораторной работе №5**

**“ Алгоритмы синхронизации процессов”**

**Аладко Анастасии Дмитриевны**

**12, МСС**

**Цель**

Изучить алгоритмы синхронизации процессов и средства операционной системы, позволяющие осуществлять синхронизацию.

**Задача**

***Вариант 1.*** *Написать диспетчер процессов, который бы постоянно держал в памяти указанное количество экземпляров заданного (командной строкой) приложения. В случае, если экземпляр приложения работает более заданного времени, то диспетчер должен принудительно завершить данный экземпляр приложения. Диспетчер должен выставлять себе приоритет выше, чем приоритет запускаемых приложений. Вывод приложений должен игнорироваться. При завершении основной программы завершить дочерние процессы*

**Описание методов**

Мои методы:  
void start\_process(const char \*program); - создаёт дочерний процесс, открывающий указанное приложение

void terminate\_process(int index); - принудительно завершает процесс, индекс которого в массиве процессов – index

void check\_processes(); - проверяет состояние процессов, если какой-то процесс закончил свою работу – выводит сообщение об этом и удаляет информацию об этом процессе, если какой-то процесс превысил лимит времени своей работы – принудительно завершает его, используя функцию terminate\_process и выводит соответствующее сообщение.

void process\_comand(char \*command); - обрабатывает команду, полученную от пользователя. Команды могут быть вида

1) start <app> <count> <Time limit>

2) exit.

1я команда запускает приложение app в количестве экземпляров count, с ограничением по времени Time limit. 2я команда завершает работу диспетчера. Команды вводится в терминале, после запуска диспетчера.

Методы из стандартных библиотек:   
perror(const char \*str) - выводит сообщение об ошибке

execlp(const char \*file, const char \*arg, ...) замещает текущий процесс новым процессом, исполняющим другую программу.

kill() отправляет сигнал процессу или группе процессов. Аргументом является идентификатор процесса (pid) и сигнал (sig), который нужно отправить.

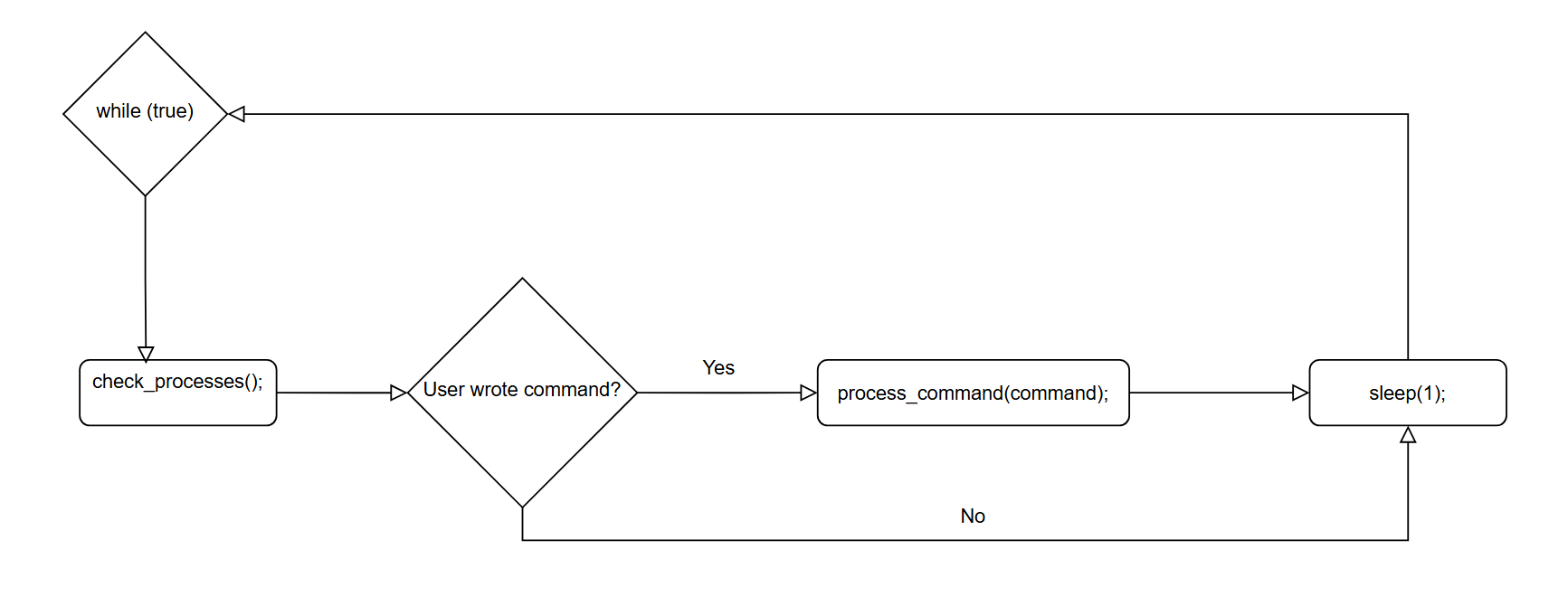
waitpid(pid\_t pid, int \*status, int options) ожидает завершения указанного процесса и возвращает его статус.

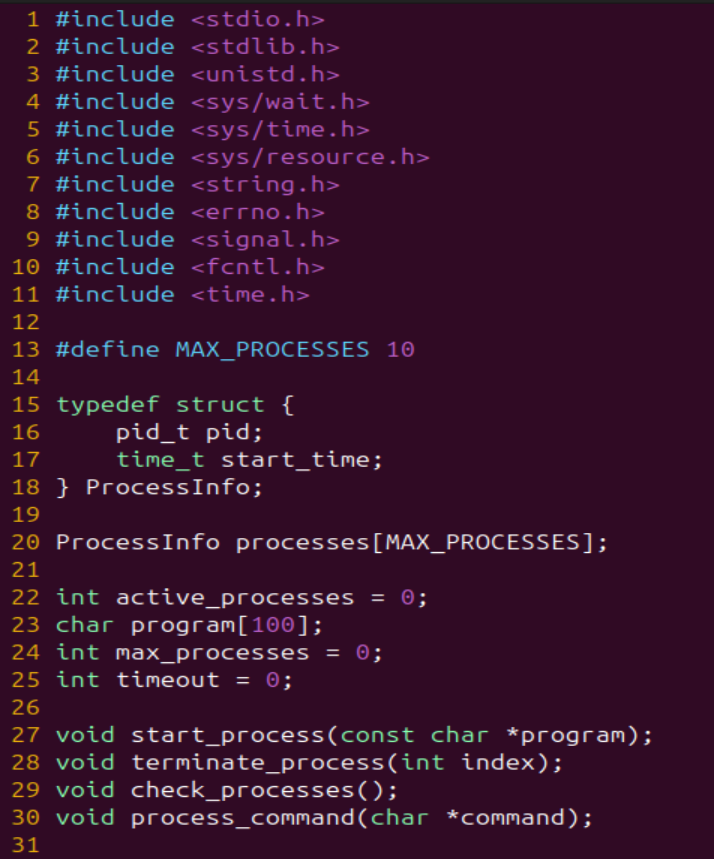
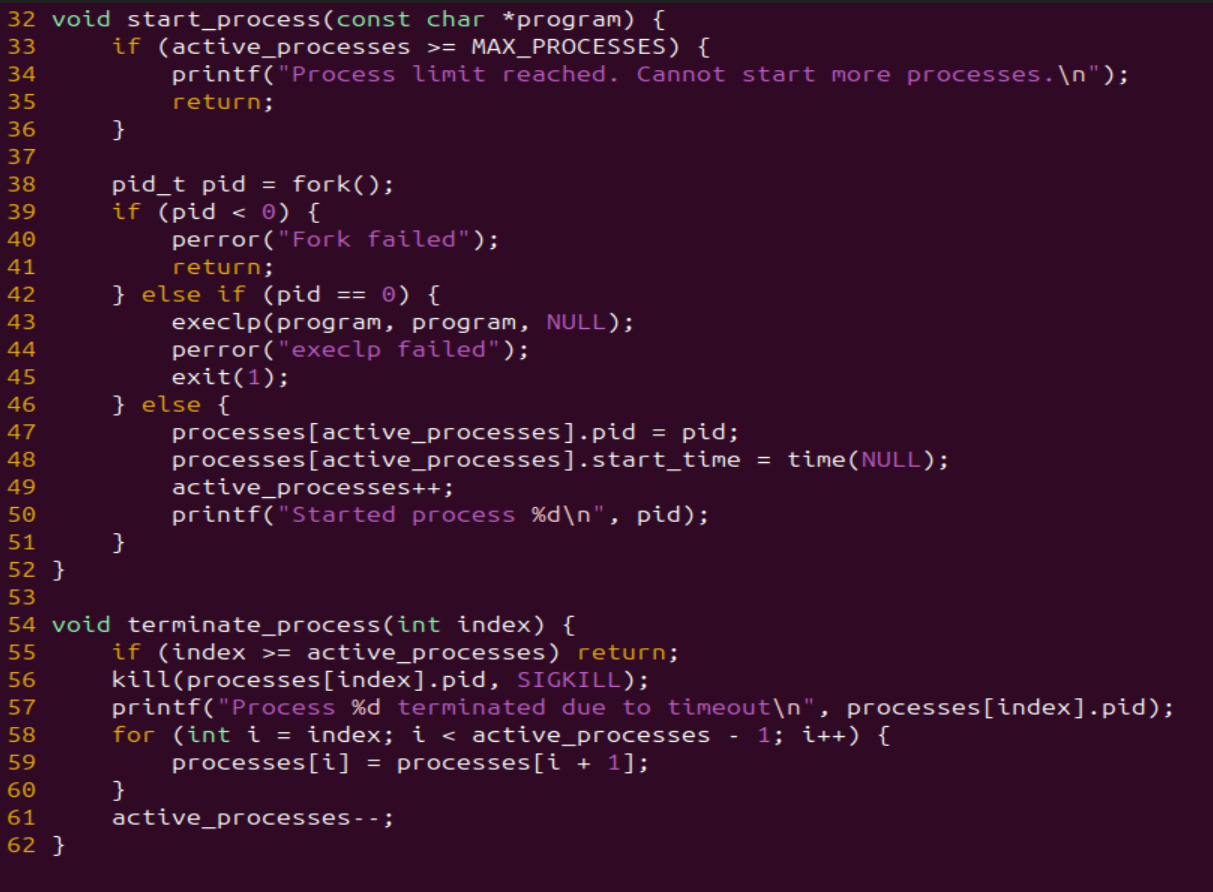
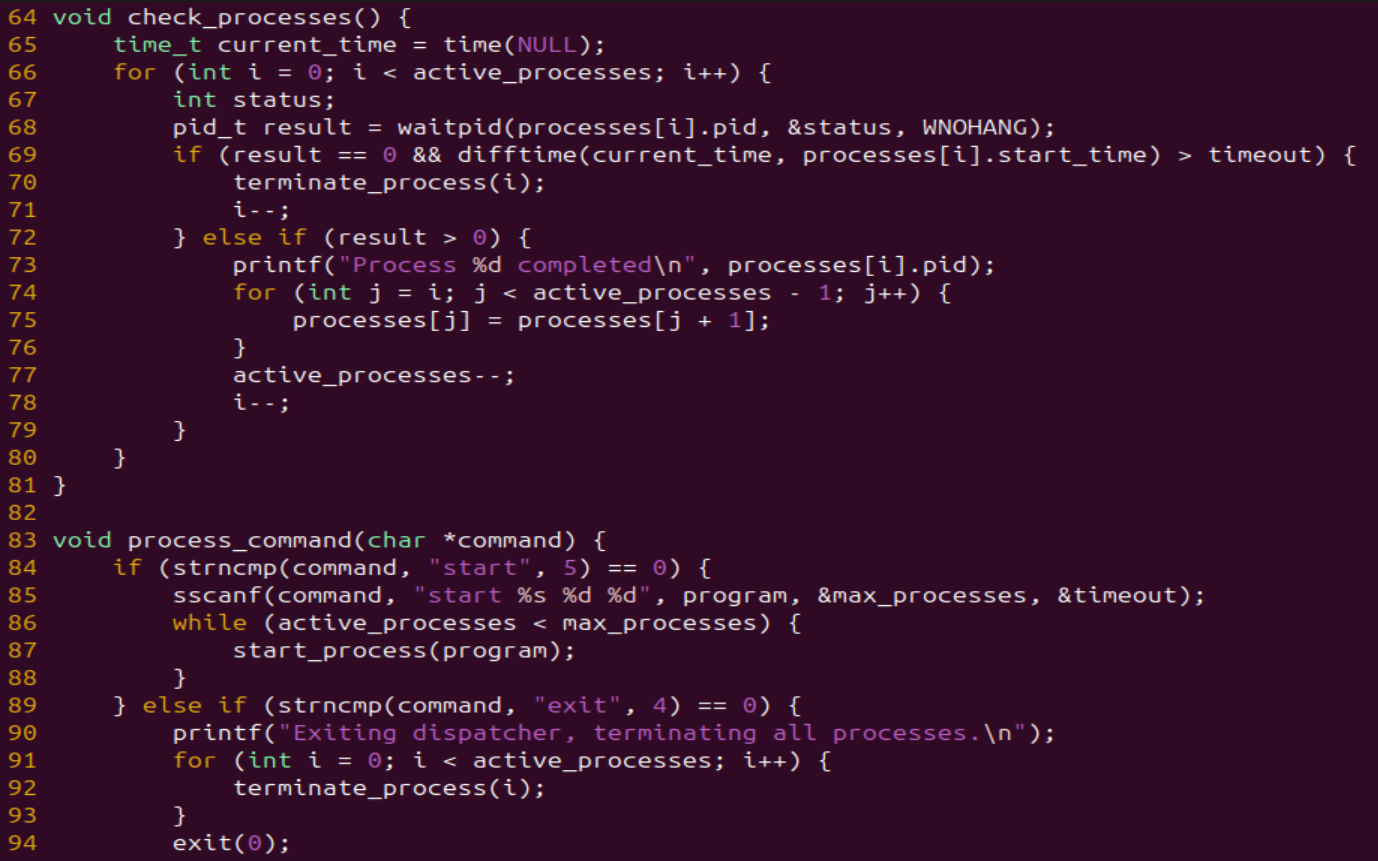
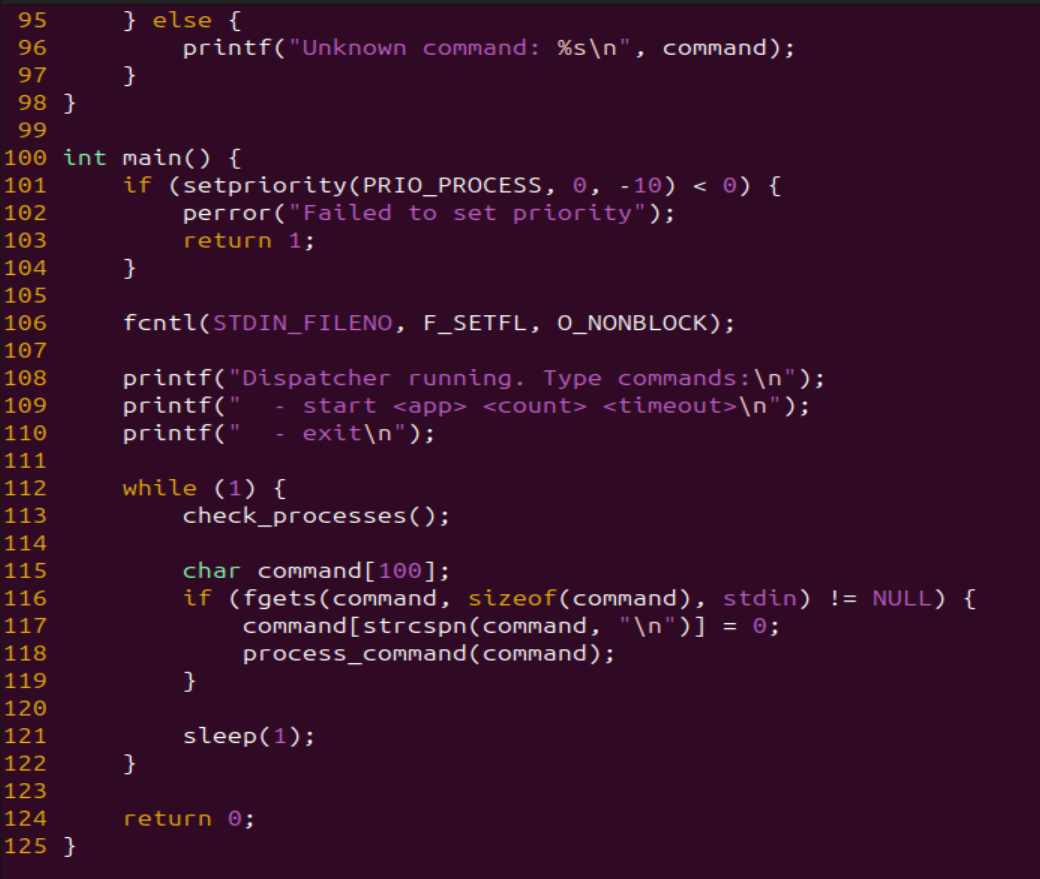
setpriority(int which, id\_t who, int prio) устанавливает приоритет процесса или группы процессов.

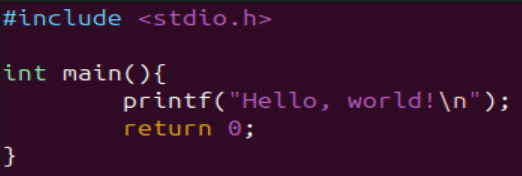
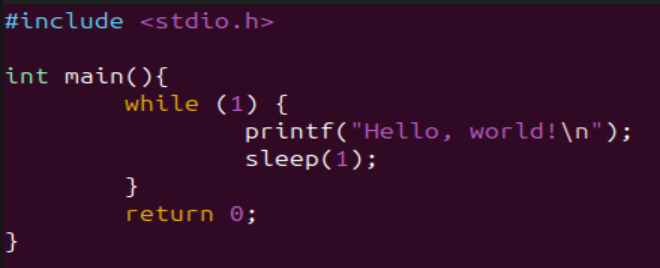
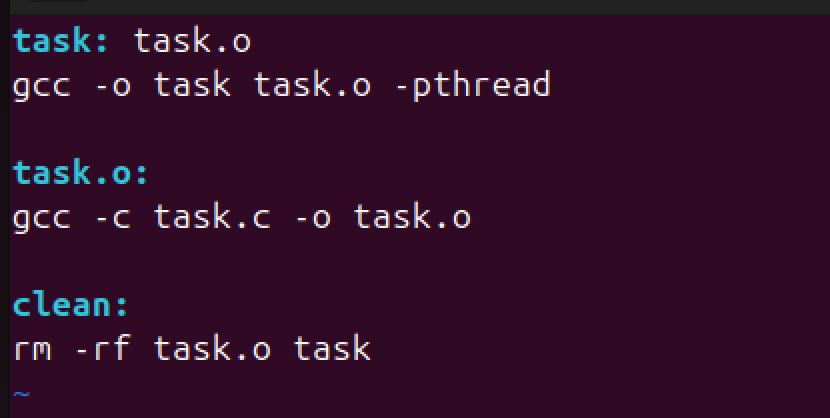
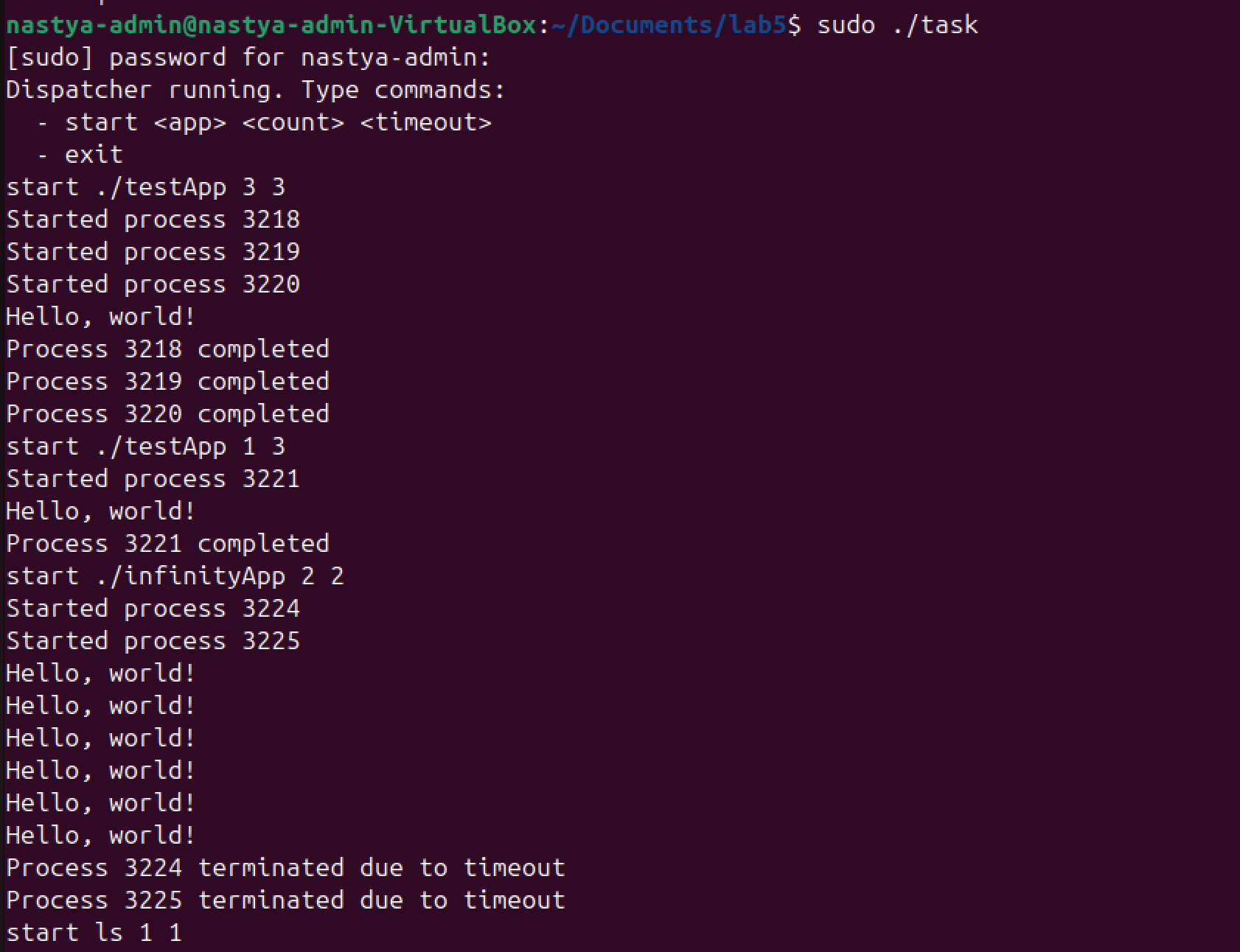
fcntl(int fd, int cmd, ...) выполняет различные операции над файловым дескриптором, я использовала fcntl(STDIN\_FILENO, F\_SETFL, O\_NONBLOCK); После этого стандартный ввод (например, с клавиатуры) будет работать в неблокирующем режиме, и программа не будет ждать ввода.

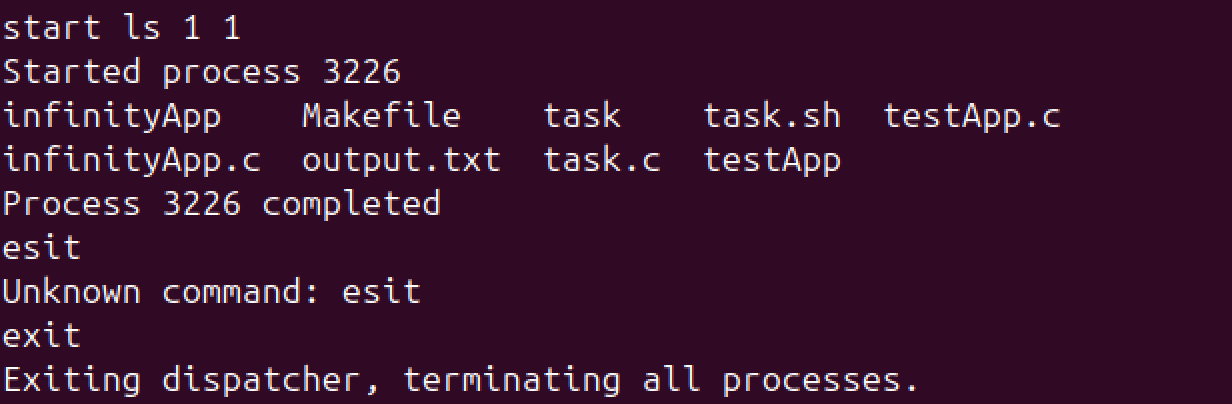
**Необходимость использования методов синхронизации**В моей задаче не было необходимости использования методов синхронизации, т.к. за работой всех процессов следит 1 единственный процесс – диспетчер, а остальные работают независимо друг от друга и не делят между собой никаких общих данных.

**Схема алгоритма решения**



Выполнение  
  
  
  


**Вспомогательные программы:**testApp:  
  
infinityApp:  
  
Makefile:  
  
**Демонстрация решения**



Как видно из демонстрации, диспетчер успешно работает: он запускает указанное количество процессов, выполняющих указанное приложение, если они успешно завершаются – выводит сообщение об их завершении, если они превышают время своей работы - принудительно завершает их и выводит сообщение об этом, также он может запускать абсолютно любые приложения, написанные и скомпилированные пользователем, или системные.

**Контрольные вопросы**

1. Поясните, в чем состоит проблема синхронизации процессов/потоков?

Это когда несколько процессов или потоков одновременно пытаются получить доступ к общим ресурсам, таким как память или файлы.

1. Что такое “эффект гонок”?

Это ситуация, когда два или более процессов или потоков обращаются к общему ресурсу и пытаются изменить его состояние, но порядок выполнения операций не контролируется.

1. Что такое клинч?

**(deadlock)**  
Ситуация, в которой два или более процессов ожидают освобождения ресурсов, которые заняты друг другом, в результате чего ни один из процессов не может продолжить выполнение.

1. Что такое взаимоисключение?

Это когда только один процесс или поток может использовать общий ресурс в данное время.

1. Что такое критическая секция с точки зрения теории операционных систем?

Это часть кода, которая обращается к общим ресурсам и должна быть выполнена атомарно, то есть без прерываний другими процессами.

1. Что такое объект ядра? Какие объекты ядра операционной системы windows или Linux вы знаете?

Это структуры данных и ресурсы, управляемые ядром операционной системы. Например процессы, потоки, семафоры, мьютексы.

1. Какие есть подходы к предотвращению, последствий тупиков?

Избегание, обнаружение, игнорирование.

1. Как можно диагностировать наличие в системе большой очереди процессов или тупика? В чем различие этих ситуаций?

Большая очередь процессов означает, что множество процессов ожидает выполнения, тогда как тупик — это ситуация, когда процессы взаимно блокируют друг друга и не могут продолжить выполнение.

1. Чем объект синхронизации критическая секция отличается от объекта синхронизации семафор в операционной системе windows.

Критическая секция позволяет только одному процессу выполнять код в критической секции одновременно. Семафор может использоваться для управления доступом к нескольким экземплярам ресурса.

1. Что такое файлы, проецируемые в память? Какие базовые механизмы операционных систем реализованы через данный подход?

Это файлы, которые отображаются в адресное пространство процесса, позволяя программе обращаться к файлу как к обычной памяти. Это позволяет ускорить доступ к данным и эффективно использовать память. Основные механизмы операционных систем, реализуемые через данный подход, включают управление памятью и файловые системы.