НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-ОБЧИСЛЮВАЛЬНИХ ТЕХНОЛОГІЙ

КАФЕДРА АВТОМАТИКИ ТА УПРАВЛІННЯ В ТЕХНІЧНИХ СИСТЕМАХ

Лабораторна робота № 4

з предмету «Програмне забезпечення комп’ютерних систем»

на тему «ПРАКТИЧНЕ ЗНАЙОМСТВО З ПРОЦЕСАМИ, ПЕРЕДАЧЕЮ ДАНИХ МІЖ ПРОЦЕСАМИ ТА ЇХ СИНХРОНІЗАЦІЯ»

Виконала:

студентка групи ІТ-з03мп

Козак О.С.

Перевірив:

Мітін С. В.

Київ – 2020

**Лабораторна робота № 4.**

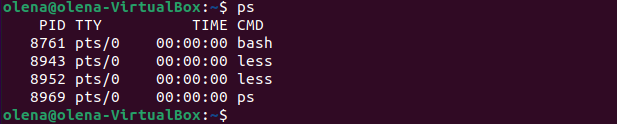
**«ПРАКТИЧНЕ ЗНАЙОМСТВО З ПРОЦЕСАМИ, ПЕРЕДАЧЕЮ ДАНИХ МІЖ ПРОЦЕСАМИ ТА ЇХ СИНХРОНІЗАЦІЯ»**

**Мета**: Вивчити базові можливості оболонки bash ОС Unix по управлінню процесами (завданнями). Розробити застосунки, які реалізують схему «клієнт» - «сервер» з використанням засобів міжпроцесної взаємодії: семафорів, розділяємої пам'яті, програмних каналів і черги повідомлень.

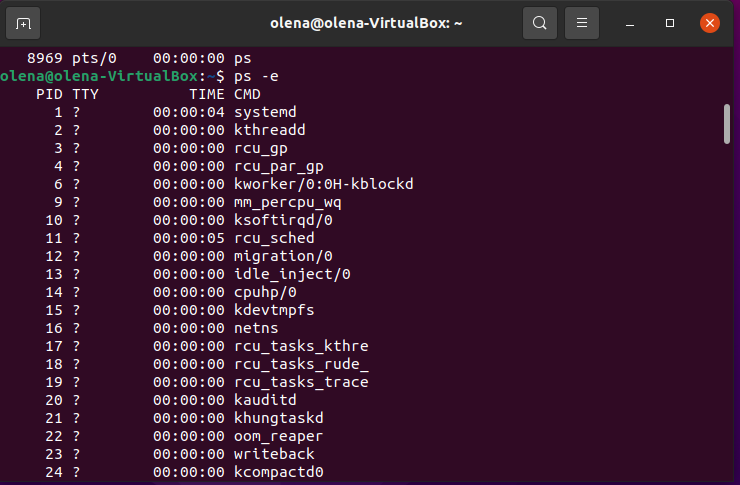
**Хід роботи:**

2. Вивчаю дію команд керування процесами:

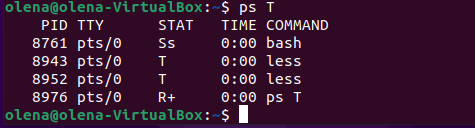
1)Команда **ps** без будь-яких параметрів і опцій показує запущені процеси, що виконуються користувачем у вікні терміналу:



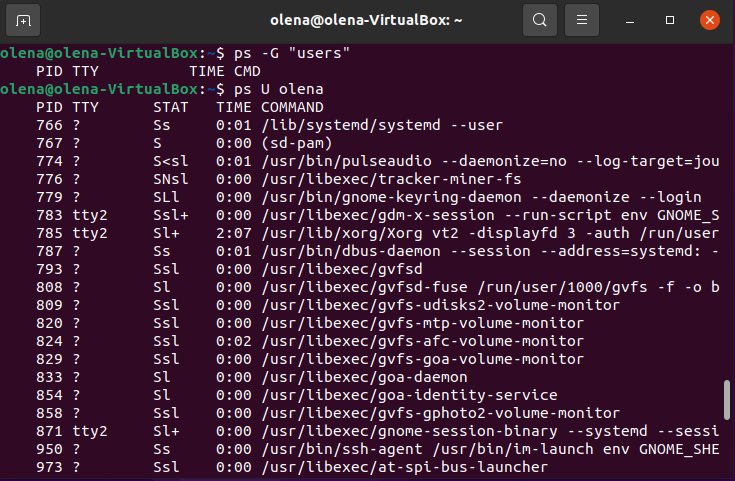
Щоб перевірити всі запущені процеси водимо команду **ps –e**:



Команда **ps T** показує нам процеси пов’язані з цим терміналом.



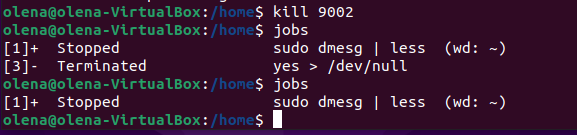
Ввівши команду **ps U <ім’я користувача**>, можу прослідкувати процеси конкретного користувача:



2) Команда **jobs** відтворює список поточник процесів. Спостерігаю виконання команд **yes** (вивід якої перенаправлений у **/del/null** (**&** - служить для запуску у фоновому режимі)) та команди **jobs**, яка надає інформацію про стан процесів. Я бачу, процес команди **yes** все ще перебуває у статусі running:

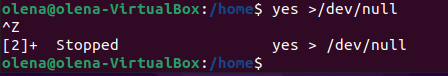


3) Команда **kill** використовується для переривання процесу:

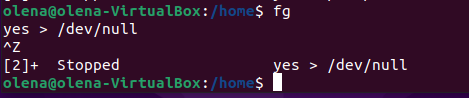


4) Команда **killall** використовує ім'я процесу замість PID і завершує всі екземпляри процесу з цим ім'ям.

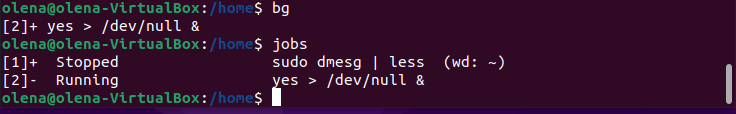
5) Для призупинення завдання треба натиснути комбінацію клавіш **Ctrl-Z:**



6) Команда **fg** використовується для відновлення процесу на передньому плані:



7) Команда **bg** запускає процес у фоновому режимі:



**3.** Забезпечую синхронізацію процесів і передачу даних між ними на прикладі двох додатків «клієнт» (рис. 1.5) і «сервер» (рис. 1.8).

На мові С++ створюєю два процеси – процес «клієнт» і процес «сервер». В якості типу переданої інформації ми використали дані, що генеруються випадковим чином. Обмін даними між процесами «клієнт» - «сервер» здійснюю з використанням іменованого програмного каналу.

1) Клієнт створює іменовані канали для зчитування та вписування інформації та надсилає серверу запит по створеному іменованому каналу для вписування.

Надсилання запиту серверу:

mkfifo**(**READ\_PIPE\_NAME**,** FILE\_MODE**);**

mkfifo**(**WRITE\_PIPE\_NAME**,** FILE\_MODE**);**

cout **<<** "Sending request to server on pipe '" **<<** WRITE\_PIPE\_NAME **<<** "'..." **<<** endl**;**

Після цього, використовуючи метод genranddata, генеруюю дані.

Метод для випадкової генерації даних:

void genranddata**(**char**\*** result**,** int size**)**

**{**

char alphanum**[]** **=** "0123456789abcdefghijklmnopqrstuvwxyz"**;**

srand**((**unsigned**)** time**(NULL)** **\*** getpid**());**

**for** **(**int i **=** 0**;** i **<** size **-** 1**;** i**++)**

**{**

result**[**i**]** **=** alphanum**[**rand**()** **%** **(sizeof(**alphanum**)** **-** 1**)];**

**}**

result**[**size **-** 1**]** **=** '\0'**;**

**}**

Відсилаюю ці дані на сервер.

Відсилання даних на сервер:

int writefd **=** open**(**WRITE\_PIPE\_NAME**,** O\_WRONLY**);**

char dataToSend**[**MESSAGE\_SIZE**];**

genranddata**(**dataToSend**,** MESSAGE\_SIZE**);**

cout **<<** "Request content: \"" **<<** dataToSend **<<** "\"" **<<** endl**;**

write**(**writefd**,** dataToSend**,** MESSAGE\_SIZE**);**

Очікуюю відповідь від сервера, отримуюю згенеровані на сервері дані та закриваю канали зв’язку. Отримання відповіді від сервера, закінчення процесу:

cout **<<** "Waiting for server response on pipe '" **<<** READ\_PIPE\_NAME **<<** "'..."**<<** endl**;**

int readfd **=** open**(**READ\_PIPE\_NAME**,** O\_RDONLY**);**

char receiveBuffer**[**MESSAGE\_SIZE**];**

ssize\_t charsRead**;**

**if** **((**charsRead **=** read**(**readfd**,** receiveBuffer**,** MESSAGE\_SIZE**))** **>** 0**)**

**{**

receiveBuffer**[**charsRead**]** **=** '\0'**;**

cout **<<** "Received server response on pipe '" **<<** READ\_PIPE\_NAME **<<** "'. "**<<** endl**;**

cout **<<** "Response content: \"" **<<** receiveBuffer **<<** "\"" **<<** endl**;**

**}**

cout **<<** "Terminating..." **<<** endl**;**

close**(**readfd**);**

close**(**writefd**);**

unlink**(**READ\_PIPE\_NAME**);**

unlink**(**WRITE\_PIPE\_NAME**);**

Код клієнта на мові С++:

#include <unistd.h>

#include <stdio.h>

#include <error.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <iostream>

#include <strings.h>

#include <fstream>

#include <sys/stat.h>

#include <errno.h>

#include <fcntl.h>

#include <string.h>

#define FILE\_MODE (S\_IRUSR | S\_IWUSR | S\_IRGRP | S\_IROTH)

#define READ\_PIPE\_NAME "/tmp/myNamedPipe1"

#define WRITE\_PIPE\_NAME "/tmp/myNamedPipe2"

#define MESSAGE\_SIZE 32

**using** **namespace** std**;**

void genranddata**(**char**\*** result**,** int size**);**

int main**(**int argc**,** char **\*\***argv**)**

**{**

mkfifo**(**READ\_PIPE\_NAME**,** FILE\_MODE**);**

mkfifo**(**WRITE\_PIPE\_NAME**,** FILE\_MODE**);**

cout **<<** "Sending request to server on pipe '"**<<**WRITE\_PIPE\_NAME**<<** "'..."**<<** endl**;**

int writefd **=** open**(**WRITE\_PIPE\_NAME**,** O\_WRONLY**);**

char dataToSend**[**MESSAGE\_SIZE**];**

genranddata**(**dataToSend**,** MESSAGE\_SIZE**);**

cout **<<** "Request content: \"" **<<** dataToSend **<<** "\"" **<<** endl**;**

write**(**writefd**,** dataToSend**,** MESSAGE\_SIZE**);**

cout **<<** "Waiting for server response on pipe '"**<<**READ\_PIPE\_NAME**<<**"'..."**<<** endl**;**

int readfd **=** open**(**READ\_PIPE\_NAME**,** O\_RDONLY**);**

char receiveBuffer**[**MESSAGE\_SIZE**];**

ssize\_t charsRead**;**

**if** **((**charsRead **=** read**(**readfd**,** receiveBuffer**,** MESSAGE\_SIZE**))** **>** 0**)**

**{**

receiveBuffer**[**charsRead**]** **=** '\0'**;**

cout **<<** "Received server response on pipe '"**<<**READ\_PIPE\_NAME**<<**"'. "**<<** endl**;**

cout **<<** "Response content: \"" **<<** receiveBuffer **<<** "\"" **<<** endl**;**

**}**

cout **<<** "Terminating..." **<<** endl**;**

close**(**readfd**);**

close**(**writefd**);**

unlink**(**READ\_PIPE\_NAME**);**

unlink**(**WRITE\_PIPE\_NAME**);**

**}**

void genranddata**(**char**\*** result**,** int size**)**

**{**

char alphanum**[]** **=** "0123456789abcdefghijklmnopqrstuvwxyz"**;**

srand**((**unsigned**)** time**(NULL)** **\*** getpid**());**

**for** **(**int i **=** 0**;** i **<** size **-** 1**;** i**++)**

**{**

result**[**i**]** **=** alphanum**[**rand**()** **%** **(sizeof(**alphanum**)** **-** 1**)];**

**}**

result**[**size **-** 1**]** **=** '\0'**;**

**}**

2) Сервер працює схожим чином. Спочатку сервер очікує на з’єднання з клієнтом по створеному клієнтом іменованому каналу та зчитує повідомлення клієнта. Зчитування повідомлення від клієнта:

cout **<<** "Waiting for client request on pipe '"**<<**READ\_PIPE\_NAME**<<**"'..." **<<** endl**;**

int readfd **=** **-**1**;**

**while** **(**readfd **==** **-**1**)**

**{**

readfd **=** open**(**READ\_PIPE\_NAME**,** O\_RDONLY**,** 0**);**

**}**

cout **<<** "Client connected on pipe '" **<<** READ\_PIPE\_NAME **<<** "'. " **<<** endl**;**

cout **<<** "Reading client request..." **<<** endl**;**

char receiveBuffer**[**MESSAGE\_SIZE**];**

ssize\_t charsRead **=** 0**;**

**if** **((**charsRead **=** read**(**readfd**,** receiveBuffer**,** MESSAGE\_SIZE**))** **>** 0**)**

**{**

receiveBuffer**[**charsRead**]** **=** '\0'**;**

cout **<<** "Request content: \"" **<<** receiveBuffer **<<** "\"" **<<** endl**;**

**}**

Після цього, з допомогою методу genranddata сервер генерує випадкові дані та надсилає відповідь клієнту, використовуючи іменований канал. Відправлення відповіді, завершення процесу:

cout **<<** "Sending response to client on pipe '" **<<** WRITE\_PIPE\_NAME **<<** "'..." **<<** endl**;**

char dataToSend**[**MESSAGE\_SIZE**];**

genranddata**(**dataToSend**,** MESSAGE\_SIZE**);**

cout **<<** "Response content: \"" **<<** dataToSend **<<** "\"" **<<** endl**;**

int writefd **=** open**(**WRITE\_PIPE\_NAME**,** O\_WRONLY**,** 0**);**

write**(**writefd**,** dataToSend**,** MESSAGE\_SIZE**);**

cout **<<** "Response sent." **<<** endl**;**

cout **<<** "Terminating..." **<<** endl**;**

close**(**readfd**);**

close**(**writefd**);**

Код серверу на мові С++ :

#include <unistd.h>

#include <stdio.h>

#include <error.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <iostream>

#include <strings.h>

#include <fstream>

#include <sys/stat.h>

#include <errno.h>

#include <fcntl.h>

#include <string.h>

#define FILE\_MODE (S\_IRUSR | S\_IWUSR | S\_IRGRP | S\_IROTH)

#define WRITE\_PIPE\_NAME "/tmp/myNamedPipe1"

#define READ\_PIPE\_NAME "/tmp/myNamedPipe2"

#define MESSAGE\_SIZE 32

**using** **namespace** std**;**

void genranddata**(**char**\*** result**,** int size**);**

int main**(**int argc**,** char **\*\***argv**)**

**{**

cout **<<** "Waiting for client request on pipe '" **<<**READ\_PIPE\_NAME**<<**"'..."**<<** endl**;**

int readfd **=** **-**1**;**

**while** **(**readfd **==** **-**1**)**

**{**

readfd **=** open**(**READ\_PIPE\_NAME**,** O\_RDONLY**,** 0**);**

**}**

cout **<<** "Client connected on pipe '" **<<** READ\_PIPE\_NAME **<<** "'. " **<<** endl**;**

cout **<<** "Reading client request..." **<<** endl**;**

char receiveBuffer**[**MESSAGE\_SIZE**];**

ssize\_t charsRead **=** 0**;**

**if** **((**charsRead **=** read**(**readfd**,** receiveBuffer**,** MESSAGE\_SIZE**))** **>** 0**)**

**{**

receiveBuffer**[**charsRead**]** **=** '\0'**;**

cout **<<** "Request content: \"" **<<** receiveBuffer **<<** "\"" **<<** endl**;**

**}**

cout **<<** "Sending response to client on pipe '"**<<**WRITE\_PIPE\_NAME**<<**"'..."**<<** endl**;**

char dataToSend**[**MESSAGE\_SIZE**];**

genranddata**(**dataToSend**,** MESSAGE\_SIZE**);**

cout **<<** "Response content: \"" **<<** dataToSend **<<** "\"" **<<** endl**;**

int writefd **=** open**(**WRITE\_PIPE\_NAME**,** O\_WRONLY**,** 0**);**

write**(**writefd**,** dataToSend**,** MESSAGE\_SIZE**);**

cout **<<** "Response sent." **<<** endl**;**

cout **<<** "Terminating..." **<<** endl**;**

close**(**readfd**);**

close**(**writefd**);**

**}**

void genranddata**(**char**\*** result**,** int size**)**

**{**

char alphanum**[]** **=** "0123456789abcdefghijklmnopqrstuvwxyz"**;**

srand**((**unsigned**)** time**(NULL)** **\*** getpid**());**

**for** **(**int i **=** 0**;** i **<** size **-** 1**;** i**++)**

**{**

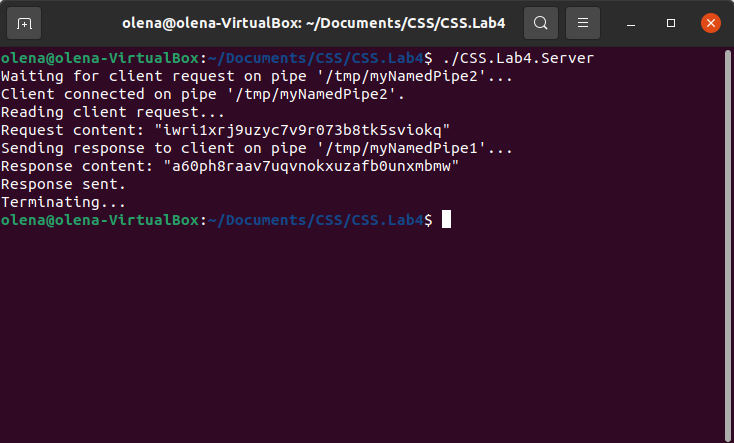
result**[**i**]** **=** alphanum**[**rand**()** **%** **(sizeof(**alphanum**)** **-** 1**)];**

**}**

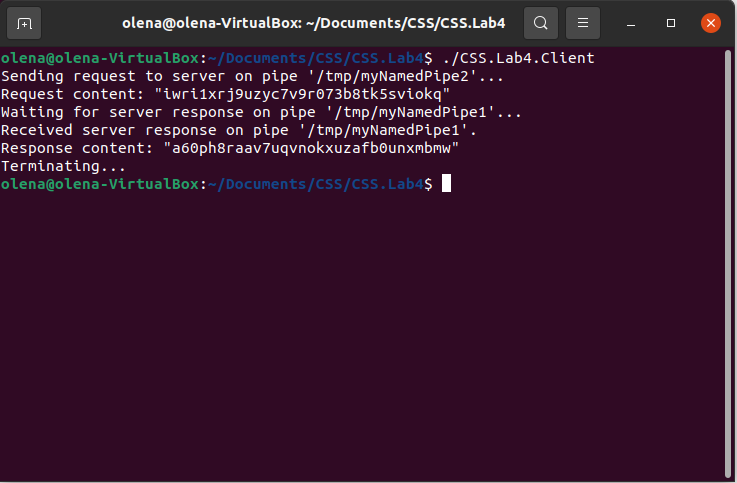
result**[**size **-** 1**]** **=** '\0'**;**

**}**

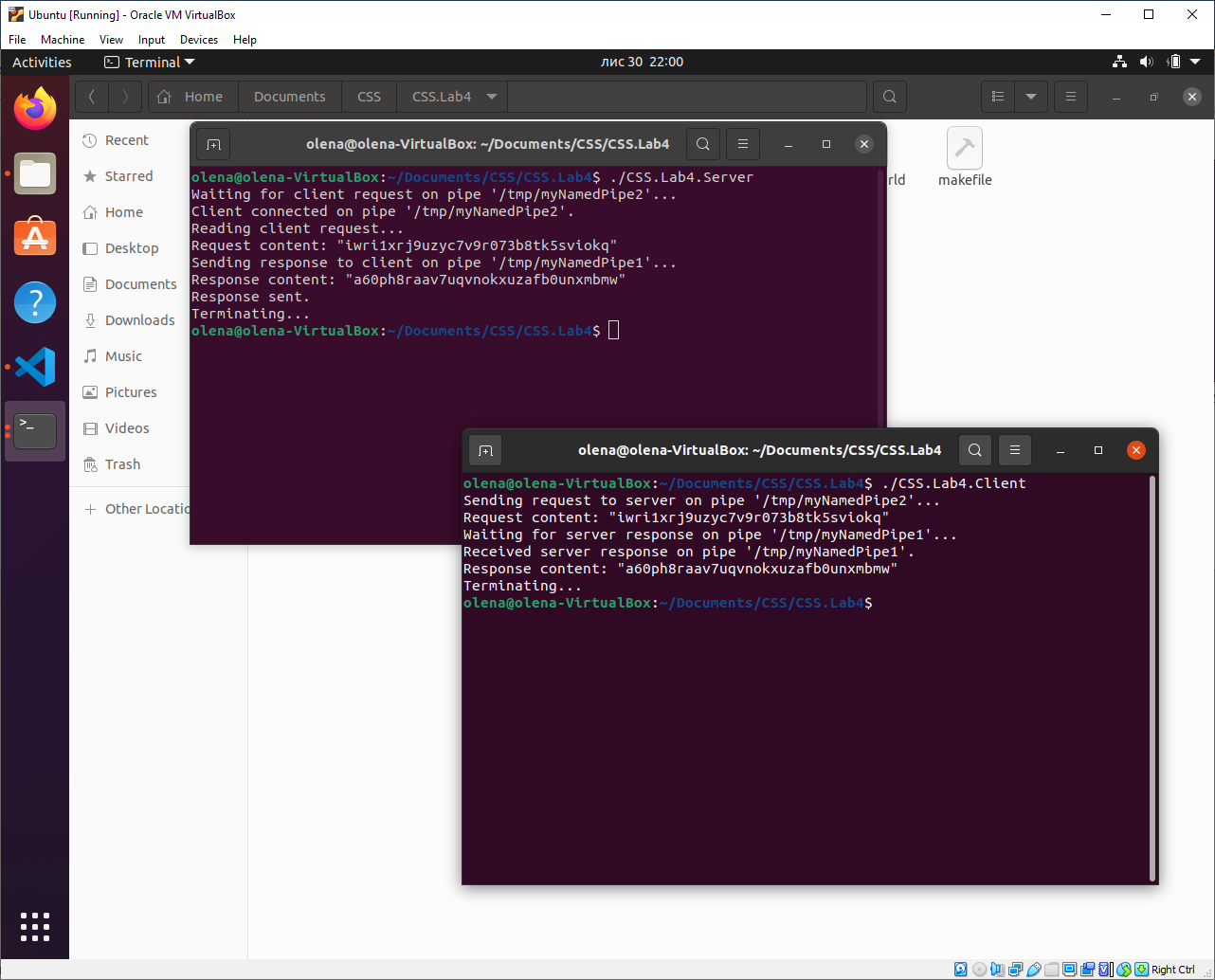
Викликаю виконуваний файл клієнта та сервера в терміналі. Спостерігаю результат після запуску сервера та клієнта .



Результат виконання програми на сервері



Результат виконання програми на клієнті



Результат виконання програми

**Висновок**:

Я вивчила базові можливості оболонки bash ОС Unix по управлінню процесами (завданнями) та розробила застосунки, які реалізують схему «клієнт» - «сервер», а також забезпечила синхронізацію процесів і передачу даних між ними на прикладі двох додатків «клієнт» та «сервер».