Université Abdelmalek Essaadi
Ecole nationale des sciences appliquées Tanger

Master Cybersécurité et Cybercriminalité
2021 – 2022

Rapport du devoir libre

Appels système en Python

Matière : Programmation système

Rendu le 20/02/2022

Réalisé par :
KHALDOUN Mohamed Amin

Encadré Par :
Mr. AMECHNOUE Khalid

Table des matières

1- Environnements de travail :

PyCharm (2021.3 Comunity Edition) :

PyCharm est un environnement de développement intégré utilisé pour programmer en Python.

Il permet l'analyse de code et contient un débogueur graphique. Il permet également la gestion des tests unitaires, l'intégration de logiciel de gestion de versions.

Dans le travail présent, je m'en suis servi pour éditer et exécuter les différents composants du projet.

Qt Designer (5.11.1):

Qt Designer est un logiciel qui permet de créer des interfaces graphiques Qt dans un environnement convivial. L'utilisateur, par glisser-déposer, place les composants d'interface graphique et y règle leurs propriétés facilement. Les fichiers d'interface graphique sont formatés en XML et portent l'extension .ui

Lors de la compilation, un fichier d'interface graphique est converti en classe Python par l'utilitaire pyuic5.

2-Appels système :

Module OS :

Le module OS en Python fournit des fonctions d'interaction avec le système d'exploitation. Le système d'exploitation relève des modules utilitaires standard de Python. Ce module fournit un moyen portable d'utiliser les fonctionnalités dépendant du système d'exploitation.

os.fork :

La méthode os.fork() en Python est utilisée pour créer un processus enfant. Cette méthode fonctionne en appelant la fonction sous-jacente du système d'exploitation fork(). Cette méthode renvoie 0 dans le processus enfant et l'ID de processus enfant dans le processus parent.

Code Pthyon	Sortie en console

import os
pid = os.fork()
if pid > 0:
  print("I am parent process:")
  print("Process ID:", os.getpid())
  print("Child's process ID:", pid)
else:
  print("\nI am child process:")
  print("Process ID:", os.getpid())
  print("Parent's process ID:", os.getppid())
	I am parent process:
	Process ID: 20900
	Child's process ID: 20901
	
	I am child process:
	Process ID: 20901
	Parent's process ID: 20900

os.system :

Exécute une commande (String) dans un sous-shell. Ceci est implémenté en appelant la fonction standard C system() et a les mêmes limitations. Les modifications apportées à sys.stdin, etc. ne sont pas reflétées dans l'environnement de la commande exécutée. Si la commande génère une sortie, elle sera envoyée au flux de sortie standard de l'interpréteur. La norme C ne spécifie pas la signification de la valeur de retour de la fonction C, donc la valeur de retour de la fonction Python dépend du système.

Code Pthyon	Sortie en console
import os os.system("date")	Sun 20 Feb 2022 09:44:56 AM EST

os.exec* :

Les méthodes os.exec* exécutent toutes un nouveau programme, remplaçant le processus actuel, elles ne renvoient pas. Sur Unix, le nouvel exécutable est chargé dans le processus actuel, et aura le même identifiant de processus (PID) que l'appelant.

Les variantes « l » et « v » des fonctions exec* différent sur la manière de passer les arguments de ligne de commande. Les variantes « l » (execl(), execlp(), execle() et execlpe()) sont probablement les plus simples à utiliser si le nombre de paramètres est fixé lors de l'écriture du code. Les paramètres individuels deviennent alors des paramètres additionnels aux fonctions exec*(). Les variantes « v » (execv(), execvp(), execve() et execvpe()) sont préférables quand le nombre de paramètres est variable et qu'ils sont passés dans une liste ou un tuple dans le paramètre args. Dans tous les cas, les arguments aux processus fils devraient commencer avec le nom de la commande à lancer, mais ce n'est pas obligatoire.

Les variantes qui incluent un « p » vers la fin (execlp(), execlpe(), execvp(), et execvpe()) utiliseront la variable d'environnement PATH pour localiser le programme file. Quand l'environnement est remplacé, le nouvel environnement est utilisé comme source de la variable d'environnement PATH. Les autres variantes execl(), execle(), execv(), et execve() n'utiliseront pas la variable d'environnement PATH pour localiser l'exécutable. path doit contenir un chemin absolue ou relatif approprié.

Pour les fonctions execle(), execlpe(), execve(), et execvpe(), le paramètre env doit être un mapping qui est utilisé pour définir les variables d'environnement du nouveau processus. Les fonctions execl(), execlp(), execv(), et execvp() causent toutes un héritage de l'environnement du processus actuel par le processus fils.

Code Pthyon	Sortie en console
import os os.execl("/bin/date", "date")	Sun 20 Feb 2022 10:24:23 AM EST
os.execlp("date", "date")	Sun 20 Feb 2022 10:26:52 AM EST
cmd = ["ls", "-l", "-n"] os.execvp("ls", cmd)	-rw-r--r-- 1 1000 1000 99 Feb 20 10:29 Execvp.py -rw-r--r-- 1 1000 1000 99 Feb 20 10:26 Execlp.py -rw-r--r-- 1 1000 1000 44 Feb 20 10:24 Execl.py

os.pipe :

Un tube est une méthode pour transmettre des informations d'un processus à un autre processus. Il n'offre qu'une communication unidirectionnelle et les informations transmises sont conservées par le système jusqu'à ce qu'elles soient lues par le processus de réception. La méthode os.pipe() en Python est utilisée pour créer un tube.

Code Pthyon	Sortie en console

import os
r, w = os.pipe()
pid = os.fork()
if pid > 0:
  os.close(r)
  print("Parent process is writing")
  text = b"Hello child process"
  os.write(w, text)
  print("Written text:", text.decode())
else:
  os.close(w)
  print("Child Process is reading")
  r = os.fdopen(r)
  print("Read text:", r.read())
	Parent process is writing
	Written text: Hello child process
	Child Process is reading
	Read text: Hello child process

os.dup / os.dup2 :

Un descripteur de fichier est une petite valeur entière qui correspond à un fichier ou à une autre ressource d'entrée / sortie, telle qu'un canal ou une socket réseau. Un descripteur de fichier est un indicateur abstrait d'une ressource et agit comme un descripteur pour effectuer diverses opérations d'E / S de niveau inférieur telles que la lecture, l'écriture, l'envoi, etc.

Par exemple: l'entrée standard est généralement un descripteur de fichier avec la valeur 0, la sortie standard est généralement un descripteur de fichier avec la valeur 1 et l'erreur standard est généralement un descripteur de fichier avec la valeur 2. Les autres fichiers ouverts par le processus actuel auront la valeur 3, 4, 5 an bientôt.

La méthode os.dup() en Python est utilisée pour dupliquer le descripteur de fichier donné. Le descripteur de fichier dupliqué n'est pas héritable, mais sur la plate-forme Windows, descripteur de fichier associé au flux standard (entrée standard: 0, sortie standard: 1, erreur standard: 2) qui peut être héritée par les processus enfants.

La méthode os.dup2() en Python est utilisée pour dupliquer un descripteur de fichier fd à une valeur donnée fd2. Le descripteur de fichier sera dupliqué dans fd2 uniquement si fd2 est disponible et si le descripteur de fichier dupliqué est héritable par défaut.

Un descripteur de fichier héritable signifie que si le processus parent a un descripteur de fichier 4 utilisé pour un fichier particulier et que le parent crée un processus enfant, le processus enfant aura également un descripteur de fichier 4 utilisé pour ce même fichier.

Code Pthyon	Sortie en console
import os fd = os.open("Files/test.txt", os.O_WRONLY) print("Original file descriptor:", fd) dup_fd = os.dup(fd) print("Duplicated file descriptor:", dup_fd) pid = os.getpid() os.system("ls -l /proc/%s/fd" %pid) os.close(fd) os.close(dup_fd) print("File descriptor duplicated successfully")	Original file descriptor: 3 Duplicated file descriptor: 4 total 0 lr-x------ 1 kali kali 64 Feb 22 11:49 0 -> pipe:[312648] l-wx------ 1 kali kali 64 Feb 22 11:49 1 -> pipe:[312649] l-wx------ 1 kali kali 64 Feb 22 11:49 2 -> pipe:[312650] l-wx------ 1 kali kali 64 Feb 22 11:49 3 -> /home/kali/PycharmProjects/ProjetProgSys/Files/test.txt l-wx------ 1 kali kali 64 Feb 22 11:49 4 -> /home/kali/PycharmProjects/ProjetProgSys/Files/test.txt File descriptor duplicated successfully

os.wait :

La méthode os.wait() en Python est utilisée par un processus pour attendre la fin d'un processus enfant.

Cette méthode renvoie un tuple contenant son PID et l'indication d'état de sortie. L'état de sortie du processus fils est indiqué par un nombre de 16 bits dont l'octet inférieur est le numéro de signal qui a tué le processus et l'octet supérieur est l'état de sortie (si le numéro de signal est zéro).

Code Pthyon	Sortie en console

import os
pid = os.fork()
if pid :
  status = os.wait()
  print("\nIn parent process-")
  print("Terminated child's process id:", status[0])
  print("Signal number that killed the child process:", status[1])
 	
else :
  print("In Child process-")
  print("Process ID:", os.getpid())
  print("Exiting")
	In Child process :
	Process ID: 24164
	Exiting
	In parent process :
	Terminated child's process id: 24164
	Signal number that killed the child process: 0

os.kill :

La méthode os.kill() est utilisée pour envoyer le signal spécifié au processus avec l'ID de processus spécifié. Les constantes des signaux spécifiques disponibles sur la plate-forme hôte sont définies dans le module de signaux.

Code Pthyon	Sortie en console

import os, signal
pid = os.fork()
if pid :
  print("In parent process :")
  os.kill(pid, signal.SIGSTOP)
  print("Signal sent, child stopped.")
  info = os.waitpid(pid, os.WSTOPPED)
  stopSignal = os.WSTOPSIG(info[1])
  print("Child stopped due to signal no:", stopSignal)
	In parent process :
	Signal sent, child stopped.
	Child stopped due to signal no: 19
	Signal name: SIGSTOP
	Signal sent, child continued.
	In child process
	Process ID: 24519
	Exitingthat killed the child process: 0

```
    print("Signal name:",
signal.Signals(stopSignal).name)
    os.kill(pid, signal.SIGCONT)
    print("Signal sent, child continued.")

else :
    print("In child process :")
    print("Process ID:", os.getpid())
    print("Exiting")
```

signal.signal :

3-Manipulation de fichiers :

fopen() :

La fonction fopen() renvoie un objet de type “file”. Cette fonction permet d’ouvrir un fichier pour y réaliser différentes opérations.

fopen() prend deux arguments : le nom du fichier à ouvrir et le mode d’ouverture (qui est par défaut r). Ce mode d’ouverture va conditionner les opérations qui vont pouvoir être faites sur le fichier par la suite. Les modes d’ouverture les plus utilisés sont les suivants :

Mode d’ouverture	Description
r	Ouvre un fichier en lecture seule. Il est impossible de modifier le fichier. Le pointeur interne est placé au début du fichier.
r+	Ouvre un fichier en lecture et en écriture. Le pointeur interne est placé au début du fichier.
a	Ouvre un fichier en écriture seule en conservant les données existantes. Le pointeur interne est placé en fin de fichier et les nouvelles données seront donc ajoutées à la fin. Si le fichier n’existe pas, le crée.
a+	Ouvre un fichier en lecture et en écriture en conservant les données existantes. Le pointeur interne est placé en fin de fichier et les nouvelles données seront donc ajoutées à la fin. Si le fichier n’existe pas, le crée.
w	Ouvre un fichier en écriture seule. Si le fichier existe, les informations existantes seront supprimées. S’il n’existe pas, crée un fichier.
w+	Ouvre un fichier en lecture et en écriture. Si le fichier existe, les informations existantes seront supprimées. S’il n’existe pas, crée un fichier.

Une fois qu’on a terminé de manipuler un fichier, il est considéré comme une bonne pratique de le fumer. Cela évite d’utiliser des ressources inutilement et d’obtenir certains comportements inattendus.

Pour fermer un fichier, on peut soit utiliser la méthode close() soit idéalement ajouter le mot clef **with** avant open() ensuite **as** lors de l'ouverture du fichier qui garantira que le fichier sera fermé automatiquement une fois les opérations terminées.

Se positionner dans un fichier :

Le curseur ou pointeur est l'endroit dans un fichier à partir duquel une opération va être faite, il indique l'emplacement à partir duquel on peut écrire ou supprimer un caractère, etc. Cela correspond par exemple à la barre clignotante dans un éditeur de texte.

Le mode d'ouverture choisi va être la première chose qui influe sur la position du pointeur. En effet, selon le mode choisi, le pointeur de fichier va se situer à une place différente. Ensuite, il faut savoir que certaines méthodes vont déplacer ce curseur lors de leur exécution, comme les méthodes de lecture du fichier par exemple.

Pour connaitre la place du pointeur interne dans un fichier et déplacer ce pointeur, nous allons pouvoir utiliser les méthodes tell() et seek().

La méthode tell() renvoie la position du pointeur interne. La méthode seek() permet de repositionner ce pointeur.

La méthode fseek() prend deux arguments : le premier argument indique de combien on souhaite décaler le curseur interne tandis que le second argument indique le point de référence à partir d'où décaler le pointeur. Ce point de référence peut être soit égal à 0 pour le début du fichier, 1 pour la position actuelle ou 2 pour la fin du fichier.

Entre autre, seek() n'accepter comme valeur de décalage que 0 ou la valeur renvoyée par tell() et le point de référence ne pout être que le début ou la fin du fichier.

write() :

Pour insérer des données dans un fichier, c'est-à-dire pour écrire dans un fichier, on utilise la méthode write(). Elle prend en argument les données à insérer, cette méthode n'accepte que des données de type chaines de caractères : on doit donc penser à convertir nos données au bon format avant tout.

De plus, les données seront écrites à partir de la position du curseur interne et si celui-ci est situé au début ou au milieu du fichier les nouvelles données écraseront les anciennes.

Finalement, la fonction write() renvoie le nombre de caractères écrits.

read() / readline() :

Pour lire entièrement un fichier, on peut utiliser la méthode read() sans argument. Cette méthode renverra le contenu du fichier sous forme de chaine de caractères.

Pour ne lire qu'une partie d'un fichier, on peut passer un nombre en argument à read() qui lui indiquera combien de caractères lire à partir de la position courante du pointeur interne.

Enfin, pour ne lire qu'une ligne d'un fichier, on peut utiliser la méthode readline().

exists() :

Pour vérifier l'existence d'un fichier, on peut utiliser la fonction exists() du module path qui appartient au module Python standard os. Cette fonction renvoie True si le chemin du fichier passé est un chemin qui existe ou False sinon.

remove() :

Pour supprimer un fichier, on peut utiliser la fonction remove() du module os. On doit passer le chemin du fichier à supprimer en argument de celle-ci.

Remarque :

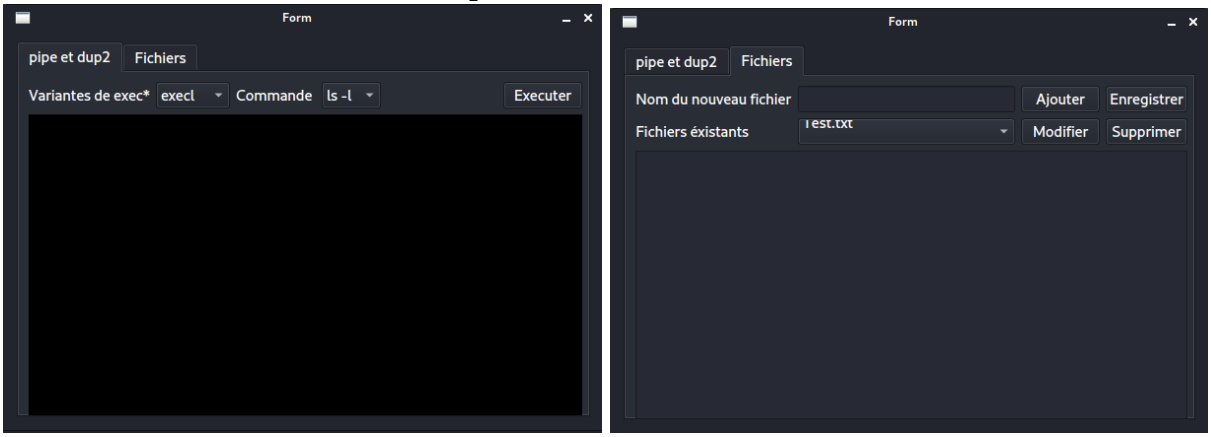
La partie pratique de cette de la manipulation de fichiers sera traité dans la section 4-Application.

4-Application :

Pout mettre en pratique les différentes méthodes et fonctions vues précédemment, j'ai réalisé une application Python avec une interface graphique mise en œuvre grâce au logiciel Qt Designer

L'application comporte deux onglets :

- La première nommée « pipe et dup2 » dans laquelle on peut exécuter différentes commandes avec une variante de la méthode os.exec* qu'on peut choisir, cette dernière est exécutée dans un processus fils dont je retransmet les sorties STDOUT et STDERR au processus père (à l'aide de la méthode os.dup2()) via un tube (créé par la méthode pipe())
- La deuxième « Fichiers » permettant de créer, modifier (via un simple éditeur de texte en bas de la fenêtre) et supprimer des fichier textes.
  Les noms de fichiers déjà existants sont chargés en profitant de l'exécution de la commande "ls" dans le répertoire Files/



Code :

Ajouter les fichiers du répertoire Files/ au comboBox :

```
def loadFilesNames(self):
        rside, wside = os.pipe()
        if not os.fork():
            # Child
            os.close(rside)
            # Make stdout go to parent
            os.dup2(wside, 1)
            # Make stderr go to parent
            os.dup2(wside, 2)
            # Execute the command
            os.execvp("ls", ["ls", "Files"])
            print("Failed to exec program!")
            sys.exit(1)
        # Parent
        os.close(wside)
        fd = os.fdopen(rside)
        files = []
        for file in fd:
            files.append(file)
        self.comboBox_Files.clear()
        self.comboBox_Files.addItems(files)
        return files
```

Évènement associé au bouton Exécuter :

```
def executerPipe(self):
        rside, wside = os.pipe()
        varExec = self.comboBox_Exec.currentText()
        cmd = self.comboBox_Cmd.currentText().split(" ")
        if not os.fork():
            # Child
            os.close(rside)
            # Make stdout go to parent
            os.dup2(wside, 1)
            # Make stderr go to parent
            os.dup2(wside, 2)
            # Execute the command
            if varExec == "execl" :
                if len(cmd)==1: os.execl("/bin/"+cmd[0], cmd[0])
                else : os.execv("/bin/"+cmd[0], cmd)
            elif varExec == "execlp" and len(cmd)==1:
                if len(cmd)==1: os.execlp(cmd[0], cmd[0])
                else: os.execvp("/bin/" + cmd[0], cmd)
            elif varExec == "execv":
                os.execv("/bin/"+cmd[0], cmd)
            elif varExec == "execvp":
                os.execvp(cmd[0], cmd)
            print("Failed to exec program!")
            sys.exit(1)
        # Parent
        os.close(wside)
        fd = os.fdopen(rside)
```

```
        lines = ""
        for line in fd:
            lines += line
        self.textBrowser_Pipe.setText(lines)
        # Prevent zombies!  Reap the child after exit
        pid, status = os.waitpid(-1, 0)
        print("Child exited: pid %d returned %d" % (pid, status))
```

Évènement associé au bouton Ajouter :

```
def createFile(self):
        fileName = self.lineEdit_NewFileName.text()
        filesNames = self.loadFilesNames()
        if fileName not in filesNames and len(fileName)>0 :
            os.system("touch Files/"+fileName+".txt")
            self.loadFilesNames()
            self.comboBox_Files.setCurrentText(fileName+".txt")
```

Évènement associé au bouton Modifier :

```
def loadFile(self):
        fileName = self.comboBox_Files.currentText().strip()
        with open("Files/"+fileName,"r") as file:
            content = file.read()
            self.plainFileTextEdit.setPlainText(content)
```

Évènement associé au bouton Enregistrer :

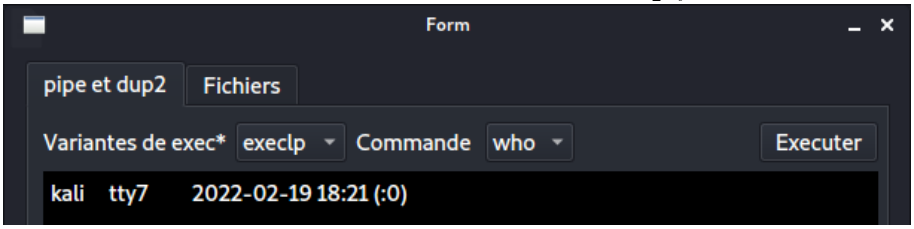
```
def saveFile(self):
        fileName = self.comboBox_Files.currentText().strip()
        with open("Files/"+fileName, "w") as file:
            content = self.plainFileTextEdit.toPlainText()
            file.write(content)
```

Évènement associé au bouton Supprimer :

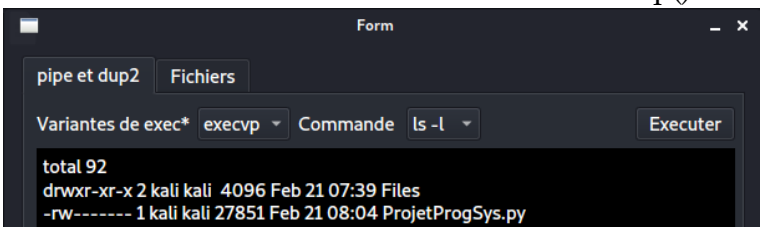
```
def deleteFile(self):
        fileName = self.comboBox_Files.currentText().strip()
        os.system("rm Files/"+fileName)
        filesNames = self.loadFilesNames()
        self.comboBox_Files.clear()
        self.comboBox_Files.addItems(filesNames)
```

Exécutions des commandes :

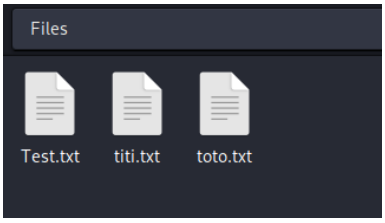
Exécuter la commande « who » avec os.execlp()



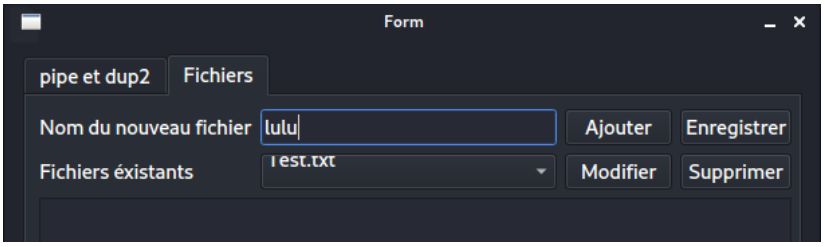
Exécuter la commande « ls -l » avec os.execvp()

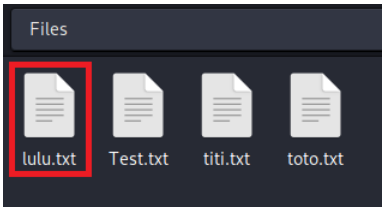


Manipulation de fichiers :

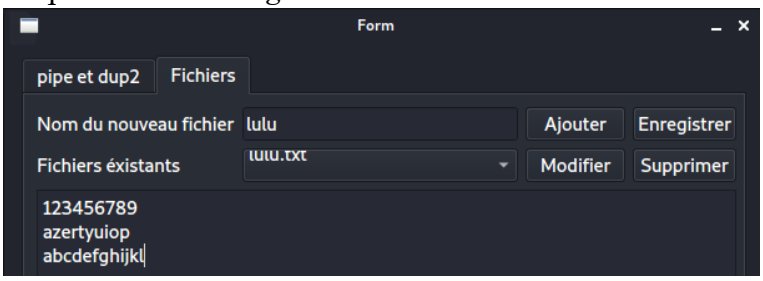


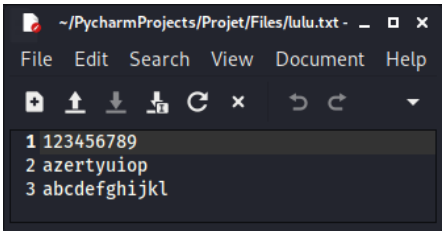
Créer un nouveau fichier :



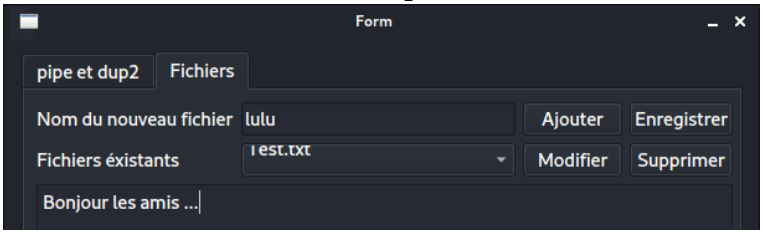


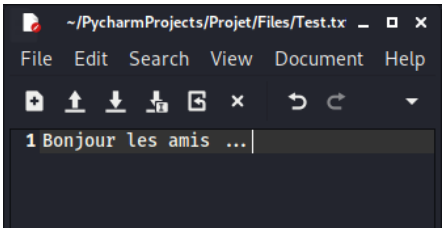
On peut modifier le fichier depuis le champ en bas de la fenêtre et puis valider en cliquant sur Enregistrer :





On choisit un autre fichier pour le modifier et on n'oublie pas d'enregistrer :





On choisit un fichier pour le supprimer :