شبیه سازی راند رابین
یک پروژه درسی از پیاده سازی های صف بندی شده

هدف پروژه:
ایجاد یک ساختار داده که بتواند فرآیندها را با استفاده از الگوریتم *Round Robin* شبیه‌سازی کند.
(این ساختار داده شامل دو تابع اصلی، startProcess و simulate است.)

توضیحات پروژه *(پروژه شامل دو تابع اصلی است)*

1. تابع startProcess:

- **هدف:** اضافه کردن یک فرآیند جدید به شبیه‌سازی.
- **ورودی:** فرآیندی که باید به شبیه‌سازی اضافه شود.

2. تابع simulate:

- **هدف:** شبیه‌سازی اجرای تمام فرآیندهای ورودی در رشته‌های جداگانه.
- **توضیحات:** تمام فرآیندها در رشته‌های جداگانه شبیه‌سازی می‌شوند و با استفاده از الگوریتم Round Robin اجرا می‌شوند. شبیه‌سازی زمانی متوقف می‌شود که هیچ فرآیند باقی‌مانده‌ای وجود نداشته باشد.

ویژگی‌های فرآیندها *(فرآیندها به عنوان یک کلاس تعریف شده و دارای ویژگی‌های زیر هستند)*
فرآیندها به عنوان یک کلاس تعریف شده و دارای ویژگی‌های زیر هستند:

- **زمان *CPU* و *I/O*:** این زمان‌ها از یک فایل با فرمت مشخص خوانده می‌شوند.
- **اولویت فرآیند:** هر فرآیند دارای سطح اولویتی است که بر زمان‌بندی تأثیر می‌گذارد.
- **زمان ورود:** زمان ورود هر فرآیند به سیستم برای شبیه‌سازی واقعی‌تر.

توضیح فایل ورودی

- **زمان CPU:** زمان‌هایی که فرآیند از CPU استفاده می‌کند. مثال: Cpu:0,7,13,28
- **زمان I/O:** زمان‌هایی که فرآیند از I/O استفاده می‌کند. مثال: IO:4,11,22

مثال دقیق) فرض کنید یک فرآیند در ثانیه 27 شروع به اجرا می‌کند و برای اولین بار در ثانیه 29 CPU را به دست می‌آورد.

گزارش‌دهی
در پایان شبیه‌سازی، گزارش‌های جامعی شامل موارد زیر ارائه شده است.

- **زمان‌های انتظار:** زمانی که هر فرآیند در صف منتظر می‌ماند.
- **زمان‌های پاسخ:** زمانی تا دریافت اولین پاسخ برای هر فرآیند.
- **زمان‌های تکمیل:** زمانی که هر فرآیند اجرای خود را به پایان می‌رساند.
- **استفاده از CPU:** درصد استفاده CPU توسط هر فرآیند.

شبیه سازی راند رابین
یک پروژه درسی از پیاده سازی های صف بندی شده

نیازمندی‌ها (موارد ملزم)

1. **زبان برنامه‌نویسی:** پایتون به دلیل سادگی و کتابخانه‌های قدرتمند توصیه می‌شود.
2. **الگوریتم زمان‌بندی:** Round Robin
3. **مدیریت رشته‌ها:** استفاده از رشته‌های جداگانه برای هر فرآیند.
4. **فایل ورودی:** خواندن و تجزیه فایل ورودی با فرمت مشخص شده.

جزئیات کلاس و شیء

- **کلاس Process:** باید با ویژگی‌هایی مانند شناسه فرآیند، زمان‌های CPU، زمان‌های I/O، اولویت، زمان ورود، زمان انتظار، زمان پاسخ، زمان تکمیل و استفاده از CPU پیاده‌سازی شود.
- **تابع startProcess:** باید یک شیء فرآیند جدید با ویژگی‌های فراهم شده ایجاد کرده و به لیست فرآیندهای شبیه‌سازی شده اضافه کند.
- **تابع simulate:** باید اجرای فرآیندها را با استفاده از الگوریتم Round Robin مدیریت کند و رشته‌های جداگانه برای هر فرآیند ایجاد کند.

بخش اضافی (اختیاری):
- بصری‌سازی بلادرنگ: افزودن بصری‌سازی بلادرنگ وضعیت فرآیندها در طول شبیه‌سازی.

تکالیف پروژه:
- کد: پیاده‌سازی کامل پروژه.
- مستندات: یک سند جامع شامل توضیحات دقیق توابع، توضیح مسئله، روش‌ها و چالش‌ها، و فرمت‌های ورودی و خروجی.

نتیجه مورد انتظار:
شبیه‌سازی باید اطمینان حاصل کند که فرآیندها به ترتیب با استفاده از الگوریتم Round Robin اجرا می‌شوند و زمان‌های CPU و I/O آنها به درستی مدیریت می‌شود.

نتیجه‌گیری:
این پروژه با هدف شبیه‌سازی فرآیندها و مدیریت زمان‌بندی آنها با استفاده از الگوریتم Round Robin طراحی شده است.

این کد یک شبیه‌ساز سیستم عامل ساده را پیاده‌سازی می‌کند که فرآیندها را مدیریت و اجرا می‌کند. این شبیه‌ساز شامل چند کلاس اصلی است که هر کدام وظایف مشخص خود را دارند. در زیر عناوینی از هر کدام از این کلاس‌ها و وظایف آنها توضیح داده شده است:

1. __SRCs.py

os, re, time, random, datetime, threading, logging, sys

ماژول‌های استاندارد پایتون برای انجام وظایف مختلف.

---

tkinter: **برای ایجاد رابط کاربری گرافیکی *(GUI).***
StringIO: **برای عملیات ورودی/خروجی رشته‌ای.**

---

quantum: **مقدار زمان برش برای زمان‌بندی پردازش‌ها.**
filename: **نام فایل ورودی.**
GUI: **یک مجموعه برای نگهداری عناصری که در *GUI* استفاده می‌شوند.**
Pid_Table: **مجموعه‌ای برای نگهداری شناسه‌های فرآیندها *(PIDs).***

کلاس برای تغییر مسیر خروجی استاندارد به یک ویجت متنی در *Tkinter* استفاده می‌شود. با استفاده از این کلاس، می‌توان متن خروجی را در ویجت متنی نمایش داد

```
__SRCs.py
import os
import re
import time
import random
import datetime
import threading
import logging
logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(message)s')
import sys
import tkinter as tk
from tkinter import ttk
from io import StringIO
quantum = 0.3  # in seconds
filename = '__Inputfile.txt'
GUI = set()
Pid_Table = set()
def pid():
    while True:
        numb = random.randint(500, 1000)  # Assuming process IDs are between 1
and 1000
        if numb not in Pid_Table:
            Pid_Table.add(numb)
            return numb
def priority():
    return random.randint(1,9)
# Function to redirect stdout to the Text widget
class TextRedirector(object):
    def __init__(self, widget, tag="stdout"):
        self.widget = widget
        self.tag = tag
    def write(self, str):
        self.widget.configure(state="normal")
        self.widget.insert("end", str, (self.tag,))
        self.widget.configure(state="disabled")
        self.widget.see("end")
    def flush(self):
        pass
def print_message(text_widget, message, tag="default"):
    text_widget.configure(state="normal")
    text_widget.insert("end", message + "\n", tag)
    text_widget.configure(state="disabled")
    text_widget.see("end")
```

این فایل شامل کد اصلی برنامه است. توابعی مانند تولید شناسه فرآیند (pid() و priority()، بازه زمانی (quantum، توابع کمکی برای تغییر خروجی استاندارد به ویجت متنی (TextRedirector) و نمایش پیام‌های مختلف در ویجت متنی (print_message) را ارائه می‌دهد.

2. PROCESS.py__

این فایل شامل کلاس PROCESS است که فرآیندها را نمایش می‌دهد. هر فرآیند شامل شناسه، اولویت، وضعیت، زمان‌های مختلف (شروع، پایان، وقفه و غیره)، و توابعی برای شبیه‌سازی اجرا و محاسبه زمان‌ها را دارد.

متد سازنده ***(Constructor: __init__)*** این متد، یک نمونه جدید از کلاس ***PROCESS*** را ایجاد کرده و ویژگی‌های مختلف آن را مقداردهی می‌کند.

متد ***run*** این متد شبیه‌سازی اجرای فرآیند را انجام می‌دهد.

متد ***Find_Behavior*** این متد، رفتار فرآیند را بر اساس زمان‌های ***CPU*** و ***IO*** تعیین می‌کند.

متد ***Quantum*** این متد، زمان‌های ***CPU*** و ***IO*** را به زمان‌بندی‌های کوتاه (کوانتا) برای زمان‌بندی گردشی تبدیل می‌کند.

متد ***print_schedule*** این متد، برنامه زمانی رفتار فرآیند را چاپ می‌کند.

متد ***print_boolActions*** این متد، اقدامات بولی فرآیند را چاپ می‌کند.

متد ***pause*** این متد، فرآیند را متوقف می‌کند.

متد ***resum*** این متد، فرآیند را از حالت توقف خارج کرده و ادامه می‌دهد.

```
# __PROCESS.py
from __SRCs import *
from __OS import *
from __IO import *
import threading
import time
import datetime

class PROCESS():
    def __init__(self, gui):
        """
        Initialize a new PROCESS instance.
        """
    self.id = pid()  # Call the pid() function to generate a unique process ID
                                                                                    ()self.priority = priority
    self.Name = f"P{self.id()}({self.priority})"
    self.thread = threading.Thread(target=self.run)
    self.pause_event = threading.Event()
    self.pause_event.set()  # Start in a paused state
    self.CPU = []
    self.IO = []
    self.behave = {}
    self.actArray = []
    self.is_complete = False
    self.start_time = 0
    self.finish_time = 0
    self.response_time = 0
    self.waiting_time = 0
    self.turnaround_time = 0
    self.paused_time = 0
    self.clock_time = 0
    self.creation_time = time.time()
    self.creation_datetime = datetime.datetime.fromtimestamp(self.creation_time).strftime('%Y-%m-%d %H:%M:%S')
    self.gui = gui
    self.quantum = quantum
    self.finish_time = 0
    self.finish_datetime = 0
    self.turn = None
    self.state = "Loading"
    self.first_print = False
    self.turn = threading.Event()  # Initialize turn as a threading Event

    def run(self):
        """
        Simulate the process execution.
        """
    def Find_Behavior(self):
        """
        Determine the process behavior based on CPU and IO bursts.
        """
    def Quantum(self):
        """
        Convert the CPU and IO bursts into time slices (quanta) for
round-robin scheduling.
        """
    def print_schedule(self):
        """
        Print the schedule of behavior for the process.
        """
    def print_boolActions(self):
        """
        Print the boolean actions of the process.
        """
    def pause(self):
        """
        Pause the process.
        """
    def resume(self):
        """
        Resume the process.
        """
```

3. __IO.py

این فایل شامل توابع مربوط به ورودی و خروجی فرآیندها است. این شامل خواندن فرآیندها از یک فایل متنی (read_processes) و نیز یک کلاس TextRedirector برای تغییر خروجی استاندارد به ویجت متنی است.

تابع *read_processes* این تابع فرآیندها را از یک فایل متنی می‌خواند و آن‌ها را به عنوان لیستی از اشیاء *PROCESS* بازمی‌گرداند. خطوط فایل بررسی می‌شوند تا زمان‌های *CPU* و *IO* هر فرآیند مشخص شود.

کلاس *TextRedirector* این کلاس برای تغییر مسیر خروجی استاندارد به یک ویجت متن در رابط کاربری گرافیکی استفاده می‌شود. این کلاس دارای متدهای *write* و *flush* است.

تابع *print_message* این تابع برای چاپ پیام‌ها در ویجت متن رابط کاربری گرافیکی استفاده می‌شود. پیام‌ها به ویجت اضافه شده و به انتهای ویجت پیمایش می‌شوند تا همیشه آخرین پیام قابل مشاهده باشد.

```
# __IO.py
from __OS import *
from __SRCs import *
from __PROCESS import *

def read_processes(filename,gui):
    processes = []
    process = None
    with open(filename, 'r') as file:
        for line in file:
            line = line.strip()
            if line.startswith("Cpu:"):
                if process:
                    processes.append(process)
                process = PROCESS(gui)
                process.CPU = list(map(int, re.findall(r'\d+', line)))
            elif line.startswith("IO:"):
                if process:
                    process.IO = list(map(int, re.findall(r'\d+', line)))
        if process:
            processes.append(process)
    return processes


class TextRedirector(object):
    def __init__(self, widget, tag="stdout"):
        self.widget = widget
        self.tag = tag
    def write(self, str):
        self.widget.configure(state="normal")
        self.widget.insert("end", str, (self.tag,))
        self.widget.configure(state="disabled")
        self.widget.see("end")
    def flush(self):
        pass


def print_message(text_widget, message, tag="default"):
    text_widget.configure(state="normal")
    text_widget.insert("end", message + "\n", tag)
    text_widget.configure(state="disabled")
    text_widget.see("end")
```

4. __OS.py

این فایل شامل کلاس OS است که عملیات مدیریت سیستم عامل شبیه‌سازی شده را ارائه می‌دهد. این شامل توابعی برای ایجاد، اضافه کردن، حذف، و اجرای فرآیندها (شامل جدول دوره روزبین)، و همچنین توابع ابزاری برای مدیریت وضعیت‌های مختلف فرآیندها (مانند ایجاد نخ‌ها برای هر فرآیند و سماق فرآیندها بر اساس اولویت) را شامل می‌شود.

متد سازنده ***Constructor: __init__()*** این متد، یک نمونه جدید از کلاس ***OS*** را ایجاد کرده و ویژگی‌های مختلف آن را مقداردهی می‌کند. متد ***create_process*** این متد برای ایجاد یک فرآیند جدید استفاده می‌شود. فرآیند ایجاد شده با استفاده از متدهای ***Find_Behavior*** و ***Quantum*** تحلیل و زمان‌بندی می‌شود. متد ***add_process*** این متد، یک فرآیند جدید را به لیست فرآیندها اضافه می‌کند. متد ***remove_process*** این متد، یک فرآیند را از لیست فرآیندها حذف می‌کند. متد ***run_round_robin*** این متد، زمان‌بندی ***Round-Robin*** را برای فرآیندها اجرا می‌کند. متد ***print_schedule*** این متد، برنامه زمانی رفتار فرآیند را چاپ می‌کند. متد ***print_boolActions*** این متد، اقدامات بولی فرآیند را چاپ می‌کند. متد ***resume*** این متد، فرآیند را از حالت توقف خارج کرده و ادامه می‌دهد. متد ***pause*** این متد، فرآیند را متوقف می‌کند. متد ***analyse*** این متد، زمان‌های ***CPU*** و ***IO*** را به طور تصادفی برای یک فرآیند جدید تعیین می‌کند. متد ***utils*** این متد، توابع کمکی را برای شبیه‌سازی سیستم عامل اجرا می‌کند. شامل دو نخ ***(Thread)*** برای چاپ اطلاعات فرآیندها و مدیریت صف فرآیندها است.	```python # __OS.py from __IO import * from __SRCs import * from __PROCESS import * class OS(): def __init__(self, gui):… def create_process(self, threshold):… def add_process(self, process):… def remove_process(self, process):… def run_round_robin(self):… def print_schedule(self, process):… def print_boolActions(self, process):… def resume(self, process):… def pause(self, process):… def analyse(self, process):… def utils(self):… ```

جمع‌بندی

کلاس OS ارائه شده در فایل __OS.py، ابزارهای لازم برای مدیریت و زمان‌بندی فرآیندها در یک سیستم عامل شبیه‌سازی شده را فراهم می‌کند. با استفاده از متدهای مختلف این کلاس می‌توان فرآیندها را ایجاد، تحلیل، و زمان‌بندی کرد و همچنین رفتار فرآیندها را مشاهده نمود.

5. __GUI.py

این فایل شامل کد برای رابط کاربری گرافیکی است. این شامل تنظیمات اولیه، ایجاد و نمایش فرآیندها در یک صفحه گرافیکی، و همچنین توابعی برای شروع و مدیریت شبیه‌سازی فرآیندها (شامل نمایش جدول زمان‌ها و خصوصیات فرآیند) را دارد. این کد مربوط به کلاس Process Execution GUI است که برای ایجاد و مدیریت یک رابط کاربری گرافیکی (GUI) برای نمایش اجرای فرآیندها در سیستم عامل شبیه‌سازی شده استفاده می‌شود. این کد از کتابخانه‌های tkinter و threading برای ایجاد رابط کاربری و مدیریت چند نخی استفاده می‌کند.

```python
# __GUI.py
from __SRCs import *
from __OS import *
from __IO import *
import tkinter as tk
from tkinter import ttk
import threading
import sys
import random
import subprocess

class ProcessExecutionGUI:
    def __init__(self, root):...

    def update_gui(self):...

    def draw_processes(self):...

    def get_color(self, process):...

    def start_process(self):...

    def Simulate(self):...

    def add_process(self):...

    def retry_processes(self):...

    def open_pdf(self):...

    def exit_application(self):...
```

پنجره اصلی *(root)* و تنظیمات آن ایجاد می‌شود.

فریم‌ها و دکمه‌های مختلف برای کنترل فرآیندها اضافه می‌شود.

یک *Text widget* برای نمایش پیام‌ها و وضعیت فرآیندها ایجاد می‌شود.

خروجی استاندارد و خطا به *Text widget* هدایت می‌شود.

لیست فرآیندها و نمونه‌ای از کلاس *OS* ایجاد می‌شود.

تابع *update_gui* برای به‌روزرسانی *GUI* به صورت دوره‌ای فراخوانی می‌شود.

متد *update_gui* این متد برای به‌روزرسانی مداوم رابط کاربری استفاده می‌شود.

متد *draw_processes* این متد برای رسم فرآیندها بر روی *canvas* استفاده می‌شو

متد *get_color* این متد رنگ مربوط به هر فرآیند را بر اساس وضعیت آن باز می‌گرداند.

متد *start_process* این متد یک نخ جدید برای شبیه‌سازی فرآیندها ایجاد و شروع می‌کند.

متد *Simulate* این متد شبیه‌سازی فرآیندها را انجام می‌دهد.

متد *add_process* این متد یک فرآیند جدید را با زمان‌بندی تصادفی ایجاد می‌کند.

متد *retry_processes* این متد برای بازنشانی شبیه‌سازی و شروع مجدد آن استفاده می‌شود.

متد *open_pdf* این متد فایل *PDF* مربوط به مستندات را باز می‌کند.

متد *exit_application* این متد برنامه را بسته و از آن خارج می‌شود.

جمع‌بندی

کلاس Process Execution GUI ارائه شده در فایل __GUI.py، ابزارهای لازم برای ایجاد و مدیریت یک رابط کاربری گرافیکی برای نمایش و مدیریت فرآیندها در یک سیستم عامل شبیه‌سازی شده را فراهم می‌کند. با استفاده از متدهای مختلف این کلاس می‌توان فرآیندها را ایجاد، شبیه‌سازی و وضعیت آن‌ها را مشاهده نمود.

6. Main.py

این فایل شامل تابع create_app() است که برنامه اصلی را ایجاد می‌کند و آن را اجرا می‌کند. این فایل بنیادی‌ترین نقطه ورود به برنامه است که شیء اصلی رابط کاربری را فراخوانی می‌کند و اجرای آن را ادامه می‌دهد.

این شبیه‌ساز از تکنولوژی‌های چون ریسه‌ها، زمان‌گذاری، ورودی/خروجی متنی، و رابط کاربری گرافیکی ترکیب شده است تا یک محیط کاربردی واقعی‌نمای سیستم عامل را شبیه‌سازی کند.

```python
####################################
#                                  #
#  @  filename    : Main.py        #
#  🚀 Developer   : Reza Khodarahimi #
#  ©  Copyright     2024           #
#                                  #
####################################

# Main.py
from __GUI import ProcessExecutionGUI
import tkinter as tk

def create_app():
    root = tk.Tk()
    app = ProcessExecutionGUI(root)
    root.mainloop()
    # while True : pass

if __name__ == '__main__':
    create_app()
```