# Multiprocessing for Image Background Removal

Course – Operating Systems
Faculty – Dr. Vallidevi K.

Submitted By :

| | |
|---|---|
| Mohammed Sahil Rizvi | 22BAI1292 |
| Mayank Raj | 22BAI1118 |
| Hardik Sondhi | 22BRS1308 |
| Aniket Verma | 22BRS1284 |

## Aim:

The aim of this project is to utilize multiprocessing, Python, and OpenCV to remove the background of multiple images at the same time. By leveraging the power of parallel processing, we enhance the efficiency of image background removal tasks, making it faster and more scalable.

## Project Elements:

1. Multiprocessing: Utilizing the processing power of multiple CPU cores to parallelize the image background removal tasks.

2. Python Programming Language: Writing the algorithms and logic for background removal using Python.

3. OpenCV Library: Leveraging OpenCV, an open-source computer vision library, for image processing tasks, including background removal.

## Background:

In the realm of image processing, removing backgrounds is a fundamental task with applications in various fields, such as photography, e-commerce, and graphic design. Traditionally, this task was computationally intensive, but with the advent of multi-core processing, Python, and OpenCV, we can achieve impressive results efficiently.

## Multiprocessing

Multiprocessing is a programming and computing paradigm that involves the simultaneous execution of multiple processes or tasks to achieve enhanced performance and efficiency. In contrast to traditional single-threaded or single-process execution, multiprocessing enables the utilization of multiple processors or cores available in modern computer systems.

At its core, multiprocessing leverages parallelism, the concept of dividing a task into smaller, independent subtasks that can be executed concurrently. This parallel execution takes place in separate processes, each with memory space, resources, and execution flow. These processes can run simultaneously on different processors or cores, allowing more efficient use of computational resources.

One of the primary motivations behind multiprocessing is the increasing prevalence of multi-core processors in modern computing architectures. Instead of relying solely on the increasing clock speeds of individual processor cores, hardware manufacturers turned to incorporating multiple cores on a single chip. This shift addressed the plateauing of single-core performance and opened the door for parallel processing as a means to boost computational capabilities.

## Multiprocessing in our project:

The code provided uses the Python multiprocessing module to achieve parallel processing, allowing multiple images to be processed concurrently. Let's break down how the code leverages multiprocessing:

1. Function to Remove Background (remove_background_from_image): The code defines a function named remove_background_from_image.This function encapsulates the logic to remove the background from a single image using the GrabCut algorithm, just like in the previous code.

2. Input and Output Directories: The input and output directories are specified. The input directory, input_dir, contains the original images that you want to process, and the output directory, output_dir, is where the processed images will be saved.

3. Creating Output Directory: The code checks if the output directory (output_dir) exists and, if not, creates it using os.makedirs(output_dir).

4. List of Image Paths and Output Paths: The code creates two lists: image_paths and output_paths. These lists contain the file paths for the input images and the corresponding output images, respectively.

5. Creating a Pool of Worker Processes: The code creates a pool of worker processes using multiprocessing.Pool(). The number of worker processes is typically equal to the number of available CPU cores, which enables concurrent image processing.

6. Preparing Input Arguments: The code prepares a list of tuples, input_output_paths, where each tuple contains the input image path and the corresponding output image path. This prepares the input data for parallel processing.

7. Parallel Image Processing: The pool.map method is used to apply the remove_background_from_image function to each image in parallel. This function takes the list of input image paths and output paths, and the pool.map method assigns each tuple to a separate worker process for concurrent execution.

8. Closing the Pool: After all image processing is completed, the pool is closed using pool.close() and then joined using pool.join(). This ensures that all processes are finished and resources are released.

## Packages used:

1. cv2 (OpenCV):
cv2, or OpenCV (Open Source Computer Vision Library), is a popular open-source library for computer vision and image processing tasks.
In the code, OpenCV is primarily used for image loading, image manipulation, and the GrabCut algorithm for background removal. It provides functions for image I/O, image processing, and computer vision tasks.

Example usage:

cv2.imread(input_image_path): Reads an image from a file.
cv2.grabCut(image, mask, rect, background_model, foreground_model, iterations, mode): Applies the GrabCut algorithm to segment an image.
cv2.imwrite(output_image_path, result): Writes an image to a file.

2. numpy (NumPy):
numpy is a fundamental package for scientific computing with Python. It provides support for working with large, multi-dimensional arrays and matrices, along with mathematical functions for array operations.
In the code, NumPy is used to create and manipulate arrays that store image data and mask information. NumPy arrays make it easier to perform element-wise operations and create binary masks.
Example usage:
np.zeros(image.shape[:2], np.uint8): Creates an array filled with zeros.
np.where(condition, x, y): Returns elements chosen from x or y based on a condition.

3. os:
The os module is part of Python's standard library and provides a portable way of using operating system-dependent functionality, such as file and directory operations.
In the code, the os module is used to manipulate file paths and create directories for input and output images.
Example usage:
os.path.join(directory, filename): Combines a directory path and a filename.
os.makedirs(directory): Creates directories, including any necessary parent directories.

4. multiprocessing:
The multiprocessing module is part of Python's standard library and allows you to create and manage multiple processes, making it suitable for parallel computing.

In the code, the multiprocessing module is used to create a pool of worker processes that concurrently process multiple images.

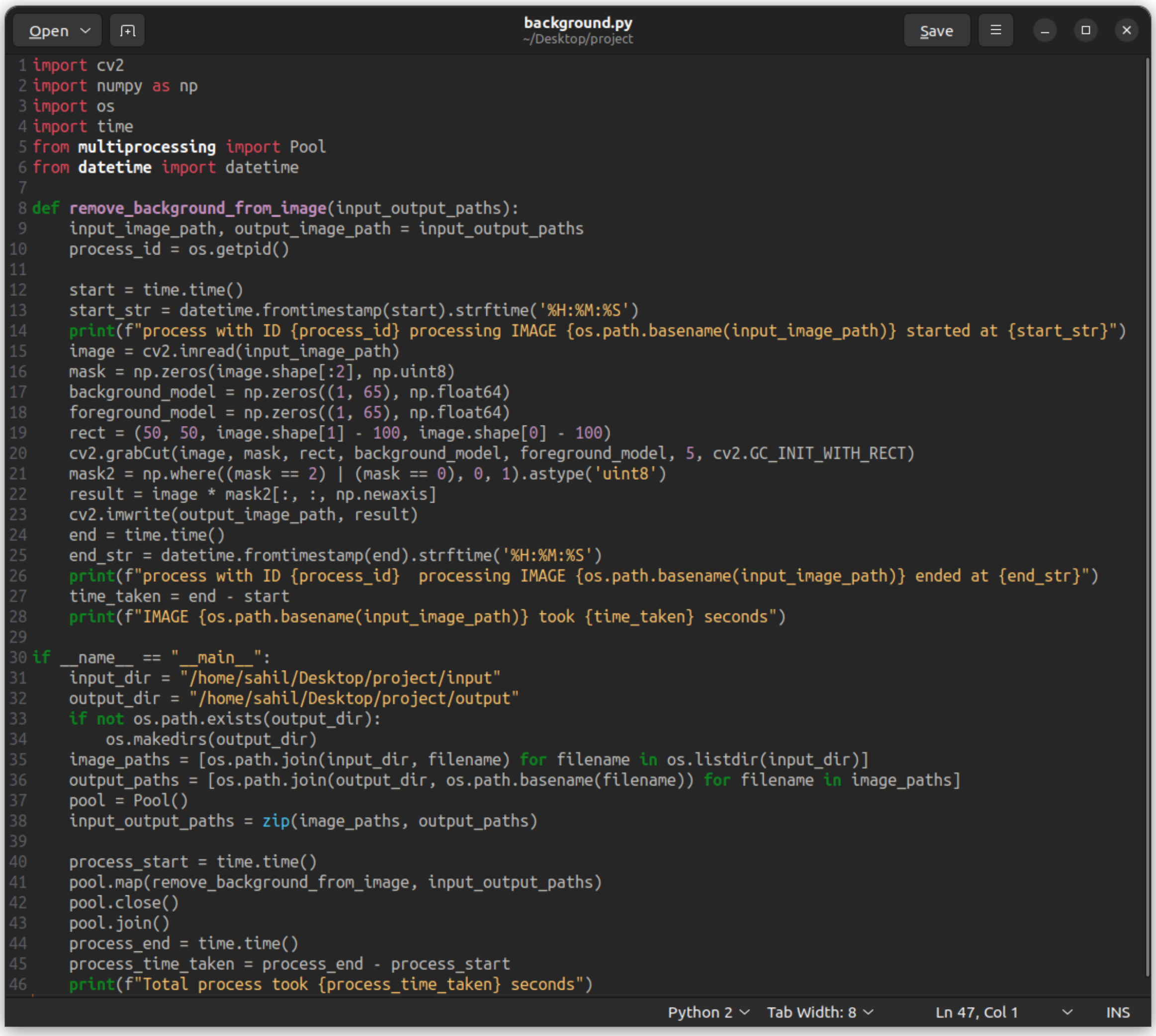
Example usage:
Pool(): Creates a pool of worker processes.
pool.map(function, iterable): Applies a function to each item in an iterable, running the function concurrently in multiple processes.
pool.close(): Closes the pool, preventing further tasks from being submitted.
pool.join(): Blocks until all processes in the pool have finished executing.

5. Time:
time.time(): This function is used to get the current time in seconds since the epoch (the epoch is a predefined point in time, often January 1, 1970). It is used to measure the elapsed time for each image-processing task.
start = time.time(): Records the start time before the image processing starts.
end = time.time(): Records the end time after the image processing is completed.
time_taken = end - start: Calculates the elapsed time for processing each image.

6. datetime module:
datetime.fromtimestamp(): Converts a timestamp (in seconds since the epoch) to a datetime object.
strftime('%H:%M:%S'): Formats the datetime object into a string with a specified format, in this case, "hours:minutes:seconds".
datetime.fromtimestamp(start).strftime('%H:%M:%S'): Converts the start timestamp to a datetime object and then formats it into a string for better human-readable output.
datetime.fromtimestamp(end).strftime('%H:%M:%S'): Converts the end timestamp to a datetime object and then formats it into a string.

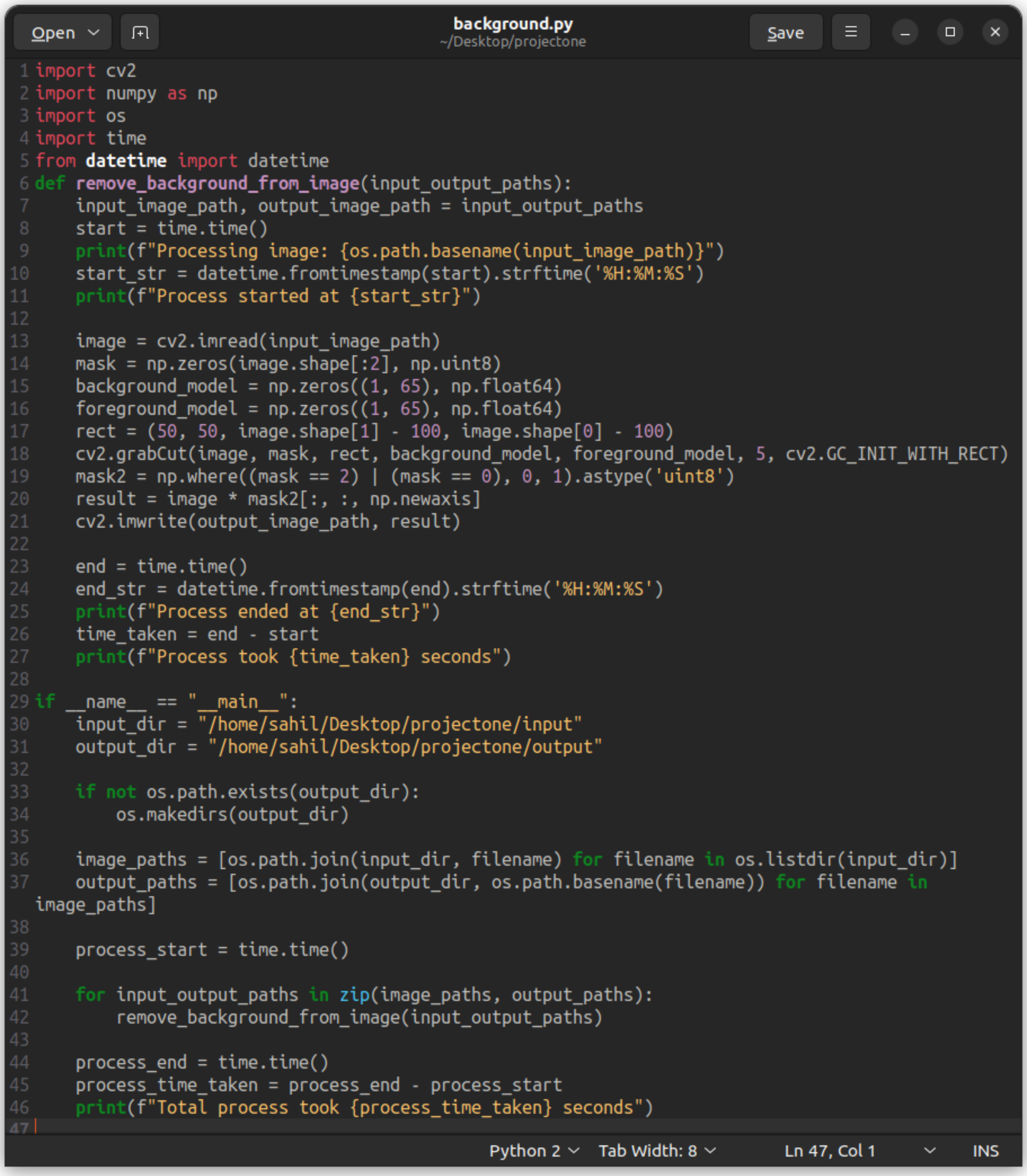
## Code:

With multiprocessing:



```python
import cv2
import numpy as np
import os
import time
from multiprocessing import Pool
from datetime import datetime

def remove_background_from_image(input_output_paths):
    input_image_path, output_image_path = input_output_paths
    process_id = os.getpid()

    start = time.time()
    start_str = datetime.fromtimestamp(start).strftime('%H:%M:%S')
    print(f"process with ID {process_id} processing IMAGE {os.path.basename(input_image_path)} started at {start_str}")
    image = cv2.imread(input_image_path)
    mask = np.zeros(image.shape[:2], np.uint8)
    background_model = np.zeros((1, 65), np.float64)
    foreground_model = np.zeros((1, 65), np.float64)
    rect = (50, 50, image.shape[1] - 100, image.shape[0] - 100)
    cv2.grabCut(image, mask, rect, background_model, foreground_model, 5, cv2.GC_INIT_WITH_RECT)
    mask2 = np.where((mask == 2) | (mask == 0), 0, 1).astype('uint8')
    result = image * mask2[:, :, np.newaxis]
    cv2.imwrite(output_image_path, result)
    end = time.time()
    end_str = datetime.fromtimestamp(end).strftime('%H:%M:%S')
    print(f"process with ID {process_id}  processing IMAGE {os.path.basename(input_image_path)} ended at {end_str}")
    time_taken = end - start
    print(f"IMAGE {os.path.basename(input_image_path)} took {time_taken} seconds")

if __name__ == "__main__":
    input_dir = "/home/sahil/Desktop/project/input"
    output_dir = "/home/sahil/Desktop/project/output"
    if not os.path.exists(output_dir):
        os.makedirs(output_dir)
    image_paths = [os.path.join(input_dir, filename) for filename in os.listdir(input_dir)]
    output_paths = [os.path.join(output_dir, os.path.basename(filename)) for filename in image_paths]
    pool = Pool()
    input_output_paths = zip(image_paths, output_paths)

    process_start = time.time()
    pool.map(remove_background_from_image, input_output_paths)
    pool.close()
    pool.join()
    process_end = time.time()
    process_time_taken = process_end - process_start
    print(f"Total process took {process_time_taken} seconds")
```

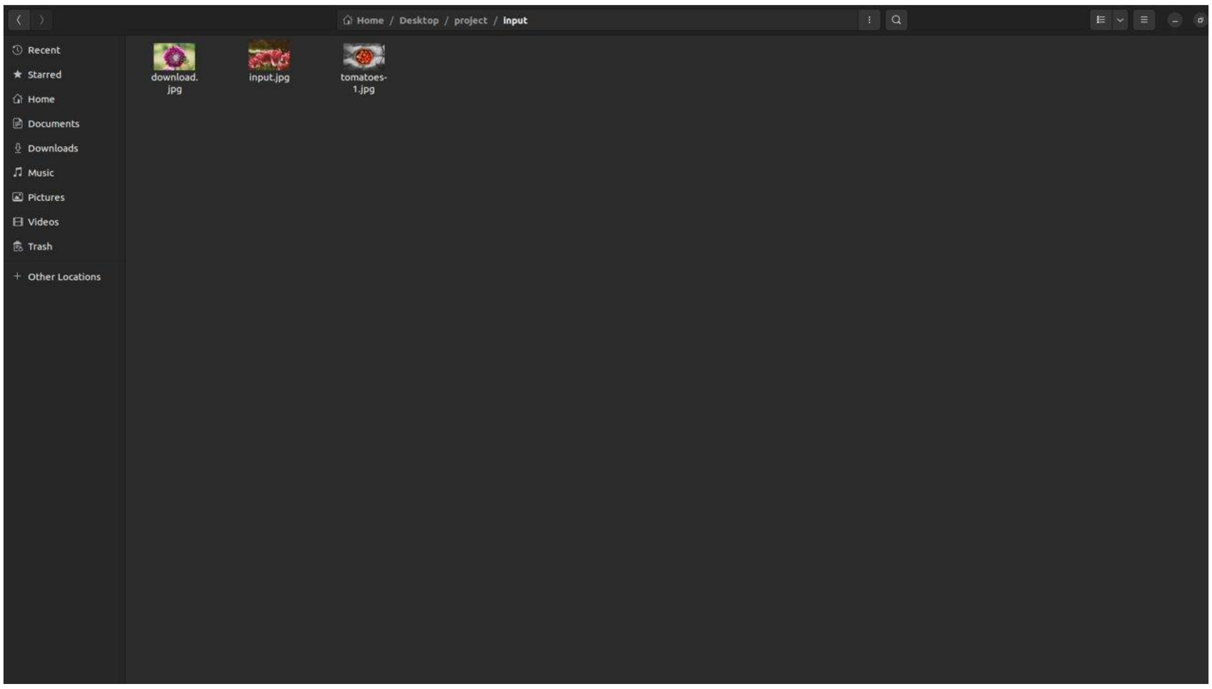
Without multiprocessing:



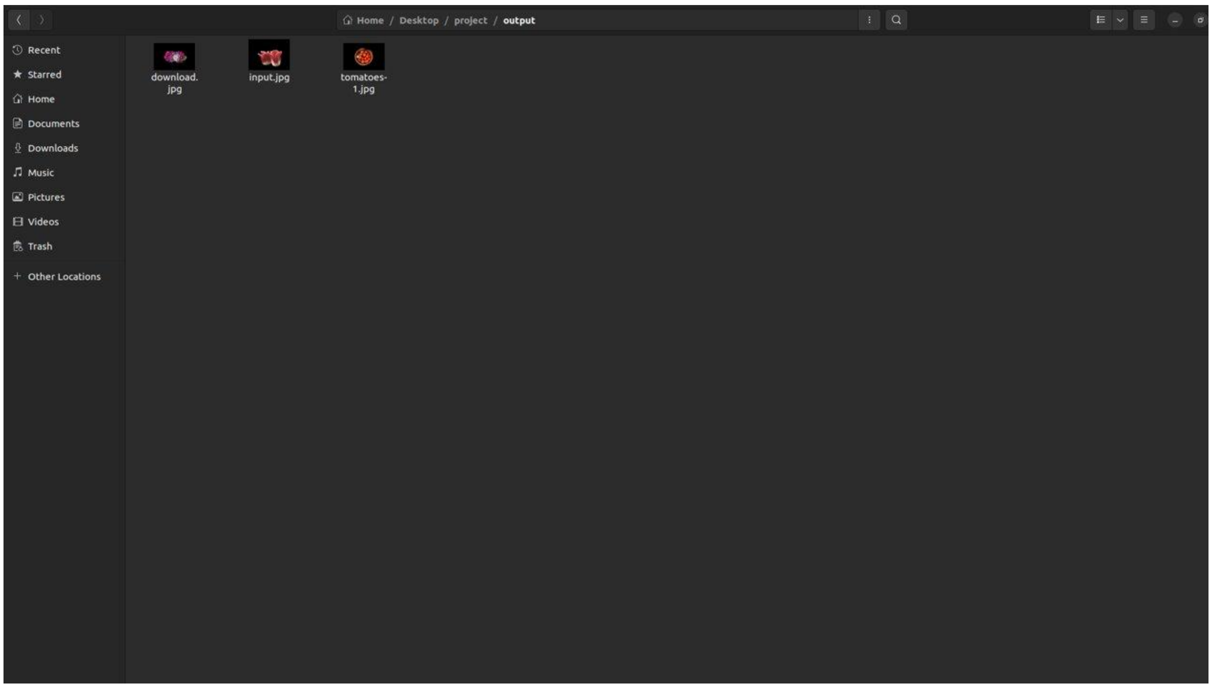
```python
import cv2
import numpy as np
import os
import time
from datetime import datetime
def remove_background_from_image(input_output_paths):
    input_image_path, output_image_path = input_output_paths
    start = time.time()
    print(f"Processing image: {os.path.basename(input_image_path)}")
    start_str = datetime.fromtimestamp(start).strftime('%H:%M:%S')
    print(f"Process started at {start_str}")

    image = cv2.imread(input_image_path)
    mask = np.zeros(image.shape[:2], np.uint8)
    background_model = np.zeros((1, 65), np.float64)
    foreground_model = np.zeros((1, 65), np.float64)
    rect = (50, 50, image.shape[1] - 100, image.shape[0] - 100)
    cv2.grabCut(image, mask, rect, background_model, foreground_model, 5, cv2.GC_INIT_WITH_RECT)
    mask2 = np.where((mask == 2) | (mask == 0), 0, 1).astype('uint8')
    result = image * mask2[:, :, np.newaxis]
    cv2.imwrite(output_image_path, result)

    end = time.time()
    end_str = datetime.fromtimestamp(end).strftime('%H:%M:%S')
    print(f"Process ended at {end_str}")
    time_taken = end - start
    print(f"Process took {time_taken} seconds")

if __name__ == "__main__":
    input_dir = "/home/sahil/Desktop/projectone/input"
    output_dir = "/home/sahil/Desktop/projectone/output"

    if not os.path.exists(output_dir):
        os.makedirs(output_dir)

    image_paths = [os.path.join(input_dir, filename) for filename in os.listdir(input_dir)]
    output_paths = [os.path.join(output_dir, os.path.basename(filename)) for filename in
image_paths]

    process_start = time.time()

    for input_output_paths in zip(image_paths, output_paths):
        remove_background_from_image(input_output_paths)

    process_end = time.time()
    process_time_taken = process_end - process_start
    print(f"Total process took {process_time_taken} seconds")
```

## Demonstration:

Input folder:



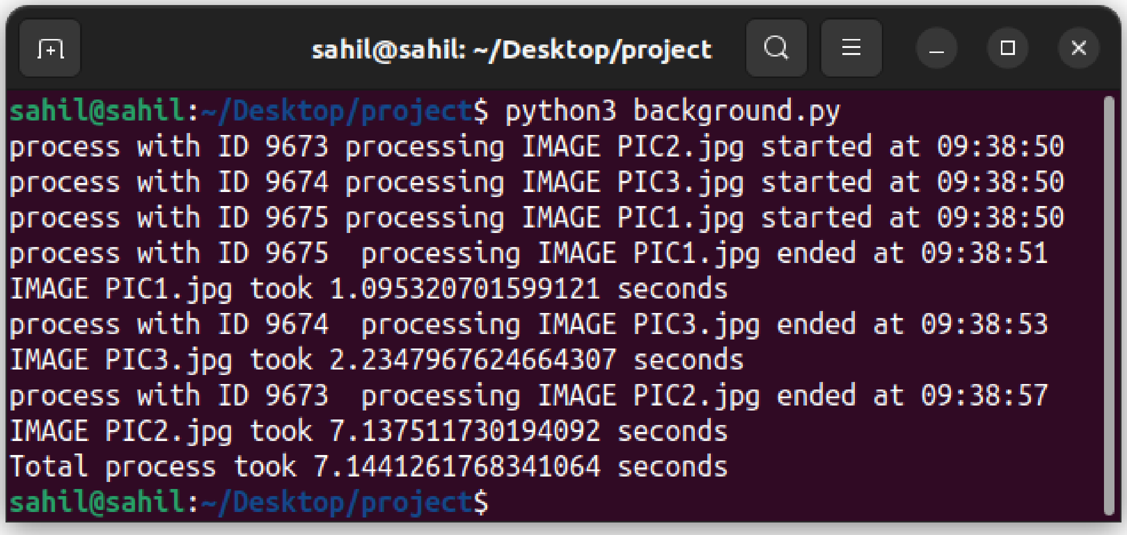
Output folder:



Photos Before and After:

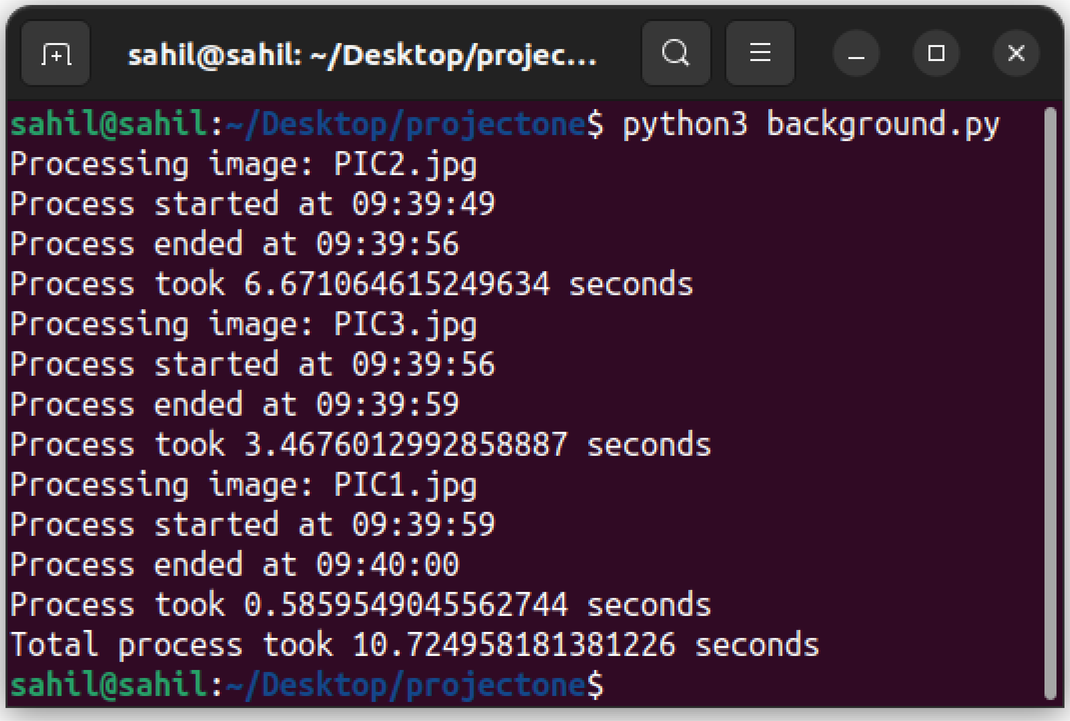
## Terminal:

With multiprocessing:



```
sahil@sahil:~/Desktop/project$ python3 background.py
process with ID 9673 processing IMAGE PIC2.jpg started at 09:38:50
process with ID 9674 processing IMAGE PIC3.jpg started at 09:38:50
process with ID 9675 processing IMAGE PIC1.jpg started at 09:38:50
process with ID 9675  processing IMAGE PIC1.jpg ended at 09:38:51
IMAGE PIC1.jpg took 1.095320701599121 seconds
process with ID 9674  processing IMAGE PIC3.jpg ended at 09:38:53
IMAGE PIC3.jpg took 2.2347967624664307 seconds
process with ID 9673  processing IMAGE PIC2.jpg ended at 09:38:57
IMAGE PIC2.jpg took 7.137511730194092 seconds
Total process took 7.1441261768341064 seconds
sahil@sahil:~/Desktop/project$
```

Without multiprocessing:



```
sahil@sahil:~/Desktop/projectone$ python3 background.py
Processing image: PIC2.jpg
Process started at 09:39:49
Process ended at 09:39:56
Process took 6.671064615249634 seconds
Processing image: PIC3.jpg
Process started at 09:39:56
Process ended at 09:39:59
Process took 3.4676012992858887 seconds
Processing image: PIC1.jpg
Process started at 09:39:59
Process ended at 09:40:00
Process took 0.5859549045562744 seconds
Total process took 10.724958181381226 seconds
sahil@sahil:~/Desktop/projectone$
```

## Conclusion:

In conclusion, this project demonstrates the power of multiprocessing combined with Python, OpenCV and Multiprocessing for efficient image background removal. The parallel processing capabilities significantly enhance the speed and performance of the task. Python's ease of use and the availability of powerful libraries like NumPy and OpenCV make it a preferred choice for image-processing tasks. By leveraging these technologies, we can achieve high-quality results in less time, revolutionizing the way we handle image processing tasks.