

Project Report

By: Mitkumar Patel

Advanced Process Manager with Process Synchronization

INTRODUCTION

The primary objective of this project is to design and implement an advanced Process Manager that not only empowers users to create, manage, and synchronize processes but also offers a comprehensive set of features to facilitate robust inter-process communication (IPC) and synchronization between threads. This Process Manager is designed to provide a unified and user-friendly interface for process creation, management, and synchronization, harnessing the capabilities of system calls for process and thread control. It stands as an embodiment of ingenuity in the domain of multi-processing, striving to enhance the performance and stability of diverse applications.

Objectives

The objectives of lab are:

1. To manipulate processes and threads in many different aspects.
2. To exploit system calls to manipulate processes and threads.
3. To develop a software which operates reliably with reducing conflicts in using system resources.

System Requirements:

Following are the system requirements of the application:

- Unix/Linux operating system (tested and programmed on Ubuntu 20)
- Python 3

Installation And Usage:

Following are the installation and usage instruction:

- To run this application, first you need the Unix/Linux operating system and then you need to install python3 using the Unix/Linux command in the terminal as : `sudo apt-get install python3`.
- To run the program in terminal, navigate to the folder using `cd` command and run the program using python by command as: `python3 Process-Manager.py`.
- Type the name of functionality given to you by the program which you want to use.

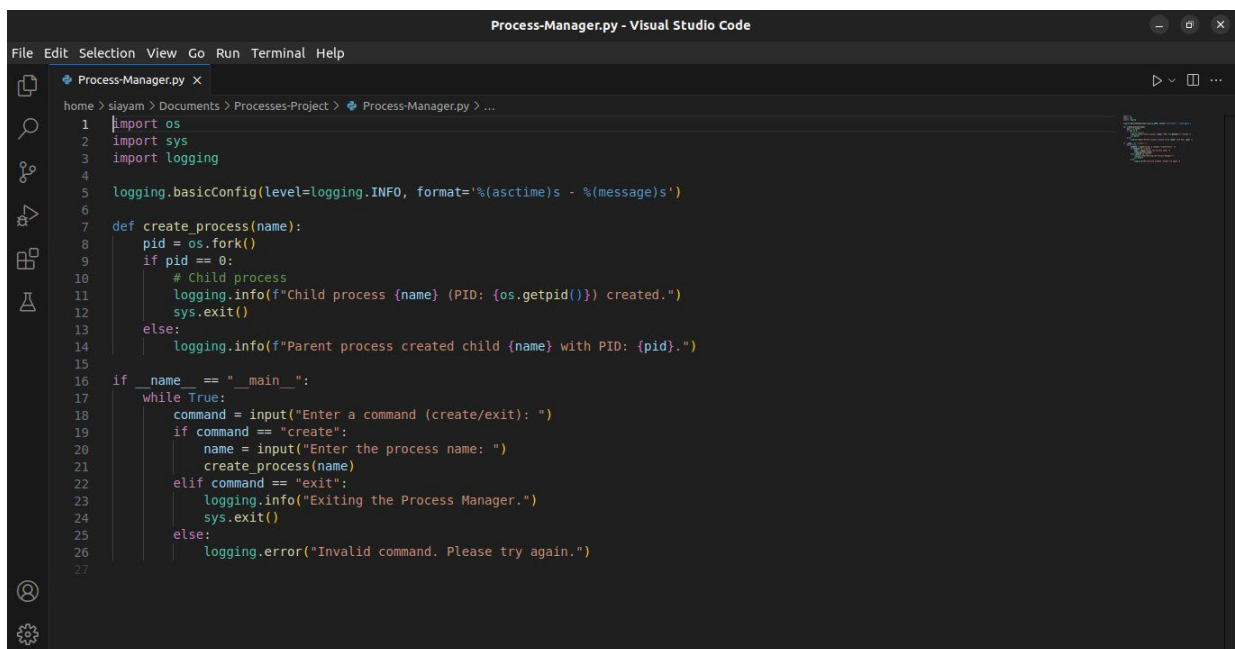
Functionalities:

❖ Process Creation

Introduction:

The system allows users to create new processes, harnessing system calls such as “ fork “ and ” exec ” This mechanism is fundamental for running multiple tasks concurrently, which is essential in scenarios ranging from server-side applications to parallel computing.

Code with explanation:



```
Process-Manager.py - Visual Studio Code
File Edit Selection View Go Run Terminal Help
Process-Manager.py x
home > siyam > Documents > Processes-Project > Process-Manager.py > ...
1 import os
2 import sys
3 import logging
4
5 logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(message)s')
6
7 def create_process(name):
8     pid = os.fork()
9     if pid == 0:
10         # Child process
11         logging.info(f"Child process {name} (PID: {os.getpid()}) created.")
12         sys.exit()
13     else:
14         logging.info(f"Parent process created child {name} with PID: {pid}.")
15
16 if __name__ == "__main__":
17     while True:
18         command = input("Enter a command (create/exit): ")
19         if command == "create":
20             name = input("Enter the process name: ")
21             create_process(name)
22         elif command == "exit":
23             logging.info("Exiting the Process Manager.")
24             sys.exit()
25         else:
26             logging.error("Invalid command. Please try again.")
27
```

This code demonstrates the process creation functionality using fork. When you run it and enter "create," it will create a child process, and you can verify this through the log output.

Output:

```
mitpatel101@Mits-Spectre:~$ cd /mnt/c/Users/patel/OneDrive\ -\ The\ Pennsylvania\ State\ University\2023\Fall\
/CMPSC\ 472/Project\ 1
mitpatel101@Mits-Spectre:/mnt/c/Users/patel/OneDrive - The Pennsylvania State University/2023/Fall/CMPSC 472/
Project 1$ python3 Process-Manager.py
Enter a command (create/list/terminate/create_thread/send_message/receive_messages/exit): create
Enter the process name: Test
2023-10-26 13:58:32,216 - Parent process created child Test with PID: 455.
Enter a command (create/list/terminate/create_thread/send_message/receive_messages/exit): 2023-10-26 13:58:32
,217 - Child process Test (PID: 455) created.
```

❖ Process listing

Introduction:

The system allows users to list all processes. It shows the PID, Name and date on which the process is created.

Code with explanation:

```
3 import logging
4
5 logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(message)s')
6
7 class ProcessManager:
8     def __init__(self):
9         self.processes = {}
10
11     def create_process(self, name):
12         pid = os.fork()
13         if pid == 0:
14             # Child process
15             logging.info(f"Child process {name} (PID: {os.getpid()}) created.")
16             sys.exit()
17         else:
18             self.processes[pid] = name
19             logging.info(f"Parent process created child {name} with PID: {pid}.")
20
21     def list_processes(self):
22         logging.info("List of running processes:")
23         for pid, name in self.processes.items():
24             logging.info(f"PID: {pid}, Name: {name}")
25
26 if __name__ == "__main__":
27     process_manager = ProcessManager()
28
29     while True:
30         command = input("Enter a command (create/list/exit): ")
31         if command == "create":
32             name = input("Enter the process name: ")
33             process_manager.create_process(name)
34         elif command == "list":
35             process_manager.list_processes()
36         elif command == "exit":
37             logging.info("Exiting the Process Manager.")
38             sys.exit()
39         else:
40             logging.error("Invalid command. Please try again.")
41
```

This code demonstrates process management, allowing you to create and list processes. Run the code, enter "create" to create a process, and enter "list" to list the running processes.

Output:

```
mitpatel101@Mits-Spectre:/mnt/c/Users/patel/OneDrive - The Pennsylvania State University/2023/Fall/CMPSC 472/Project 1$ cd /mnt/c/Users/patel/OneDrive\ -\ The\ Pennsylvania\ State\ University/2023/Fall/CMPSC\ 472/Project 1
mitpatel101@Mits-Spectre:/mnt/c/Users/patel/OneDrive - The Pennsylvania State University/2023/Fall/CMPSC 472/Project 1$ python3 Process-Manager.py
Enter a command (create/list/terminate/create_thread/send_message/receive_messages/exit): list
2023-10-26 14:01:15,840 - List of running processes:
Enter a command (create/list/terminate/create_thread/send_message/receive_messages/exit): create
Enter the process name: Test1
2023-10-26 14:01:29,570 - Parent process created child Test1 with PID: 465.
Enter a command (create/list/terminate/create_thread/send_message/receive_messages/exit): 2023-10-26 14:01:29,571 - Child process Test1 (PID: 465) created.
create
Enter the process name: Test2
2023-10-26 14:01:34,839 - Parent process created child Test2 with PID: 466.
Enter a command (create/list/terminate/create_thread/send_message/receive_messages/exit): 2023-10-26 14:01:34,839 - Child process Test2 (PID: 466) created.
list
2023-10-26 14:01:38,694 - List of running processes:
2023-10-26 14:01:38,694 - PID: 465, Name: Test1
2023-10-26 14:01:38,694 - PID: 466, Name: Test2
Enter a command (create/list/terminate/create_thread/send_message/receive_messages/exit):
```

❖ Process Terminating

Introduction:

The system allows the user to terminate any processes that are created by the user.

Code with explanation:

```
8 class ProcessManager:
9     def __init__(self):
10         self.processes = {}
11
12     def create_process(self, name):
13         pid = os.fork()
14         if pid == 0:
15             # Child process
16             logging.info(f"Child process {name} (PID: {os.getpid()}) created.")
17             time.sleep(5) # Simulate process execution
18             sys.exit()
19         else:
20             self.processes[pid] = name
21             logging.info(f"Parent process created child {name} with PID: {pid}.")
22
23     def list_processes(self):
24         logging.info("List of running processes:")
25         for pid, name in self.processes.items():
26             logging.info(f"PID: {pid}, Name: {name}")
27
28     def terminate_process(self, pid):
29         if pid in self.processes:
30             os.kill(pid, 9) # Kill the process
31             del self.processes[pid]
32             logging.info(f"Process with PID {pid} terminated.")
33         else:
34             logging.error(f"Process with PID {pid} not found.")
35
36 if __name__ == "__main__":
37     process_manager = ProcessManager()
38
39     while True:
40         command = input("Enter a command (create/list/terminate/exit): ")
41         if command == "create":
42             name = input("Enter the process name: ")
43             process_manager.create_process(name)
44         elif command == "list":
45             process_manager.list_processes()
46         elif command == "terminate":
47             pid = int(input("Enter the PID of the process to terminate: "))
48             process_manager.terminate_process(pid)
49         elif command == "exit":
50             logging.info("Exiting the Process Manager.")
51             sys.exit()
52         else:
53             logging.error("Invalid command. Please try again.")
54
```

This code demonstrates process management, allowing you to create, list and terminate processes. Run the code, enter "create" to create a process, and enter "list" to list the running processes and terminate to kill the process by providing it PID.

Output:

```
mitpatel101@Mits-Spectre:/mnt/c/Users/patel/OneDrive - The Pennsylvania State University/2023/Fall/CMPSC 472/Project 1$ cd /mnt/c/Users/patel/OneDrive - The Pennsylvania State University/2023/Fall/CMPSC 472/Project 1
mitpatel101@Mits-Spectre:/mnt/c/Users/patel/OneDrive - The Pennsylvania State University/2023/Fall/CMPSC 472/Project 1$ python3 Process-Manager.py
Enter a command (create/list/terminate/create_thread/send_message/receive_messages/exit): list
2023-10-26 14:01:15,840 - List of running processes:
Enter a command (create/list/terminate/create_thread/send_message/receive_messages/exit): create
Enter the process name: Test1
2023-10-26 14:01:29,570 - Parent process created child Test1 with PID: 465.
Enter a command (create/list/terminate/create_thread/send_message/receive_messages/exit): 2023-10-26 14:01:29,571 - Child process Test1 (PID: 465) created.
create
Enter the process name: Test2
2023-10-26 14:01:34,839 - Parent process created child Test2 with PID: 466.
Enter a command (create/list/terminate/create_thread/send_message/receive_messages/exit): 2023-10-26 14:01:34,839 - Child process Test2 (PID: 466) created.
list
2023-10-26 14:01:38,694 - List of running processes:
2023-10-26 14:01:38,694 - PID: 465, Name: Test1
2023-10-26 14:01:38,694 - PID: 466, Name: Test2
Enter a command (create/list/terminate/create_thread/send_message/receive_messages/exit): create
Enter the process name: Test3
2023-10-26 14:08:44,298 - Parent process created child Test3 with PID: 552.
Enter a command (create/list/terminate/create_thread/send_message/receive_messages/exit): 2023-10-26 14:08:44,299 - Child process Test3 (PID: 552) created.
list
2023-10-26 14:08:46,715 - List of running processes:
2023-10-26 14:08:46,715 - PID: 465, Name: Test1
2023-10-26 14:08:46,715 - PID: 466, Name: Test2
2023-10-26 14:08:46,716 - PID: 552, Name: Test3
Enter a command (create/list/terminate/create_thread/send_message/receive_messages/exit): terminate
Enter the PID of the process to terminate: 465
2023-10-26 14:08:58,111 - Process with PID 465 terminated.
Enter a command (create/list/terminate/create_thread/send_message/receive_messages/exit): list
2023-10-26 14:09:02,177 - List of running processes:
2023-10-26 14:09:02,177 - PID: 466, Name: Test2
2023-10-26 14:09:02,177 - PID: 552, Name: Test3
Enter a command (create/list/terminate/create_thread/send_message/receive_messages/exit):
```

❖ Thread Support:

Introduction:

It leverages system calls like “pthread_create” and synchronization primitives to enhance parallel execution, providing a solid foundation for multi-threaded applications..

Code with explanation:

```
home > siyam > Documents > Processes-Project > ➡ thread-check.py > ...
1  import threading
2  import logging
3  import time
4
5  logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(message)s')
6
7  class ThreadManager:
8      def create_thread(self, thread_name):
9          thread = threading.Thread(target=self.thread_function, args=(thread_name,))
10         thread.start()
11         logging.info(f"Thread {thread_name} started.")
12
13         def thread_function(self, thread_name):
14             logging.info(f"Thread {thread_name} is doing some work.")
15             time.sleep(3)
16             logging.info(f"Thread {thread_name} finished its work.")
17
18  if __name__ == "__main__":
19      thread_manager = ThreadManager()
20
21      while True:
22          command = input("Enter a command (create/exit): ")
23          if command == "create":
24              thread_name = input("Enter the thread name: ")
25              thread_manager.create_thread(thread_name)
26          elif command == "exit":
27              logging.info("Exiting the Thread Manager.")
28              break
29          else:
30              logging.error("Invalid command. Please try again.")
31
```

This code demonstrates the threading functionality, allowing you to create thread to run multiple processes side by side.

❖ IPC Functionality:

Introduction:

Inter-Process Communication (IPC) allows processes to communicate and share data. One common method for IPC is using pipes.

Code with explanation:

```
95
96     def send_message(self, source_pid, target_pid, message):
97         if source_pid in self.processes and target_pid in self.processes:
98             source_process = self.processes[source_pid]
99             target_process = self.processes[target_pid]
100             with target_process.lock: # Mutex (Lock) for process-level synchronization
101                 target_process.message_queue.put(f"From {source_process.name}: {message}")
102         else:
103             logging.error("Invalid source or target PID.")
104
105     def receive_messages(self, pid):
106         if pid in self.processes:
107             process = self.processes[pid]
108             while not process.message_queue.empty():
109                 message = process.message_queue.get()
110                 logging.info(f"Process {process.name} received message: {message}")
111         else:
112             logging.error("Invalid PID.")
113
```

This code demonstrates the Inter-Process Communication (IPC) functionality, allowing processes to send and receive messages from one another and communicate with each other.

Output:

```
mitpatel101@Mits-Spectre:~$ cd /mnt/c/Users/patel/OneDrive\ -\ The\ Pennsylvania\ State\ Univ
ersity\2023\Fall\CMPSC\ 472\Project\ 1
mitpatel101@Mits-Spectre:/mnt/c/Users/patel/OneDrive - The Pennsylvania State University/2023
/Fall/CMPSC 472/Project 1$ python3 Process-Manager.py
Enter a command (create/list/terminate/create_thread/send_message/receive_messages/exit): c
create
Enter the process name: test1
2023-10-27 14:46:37,694 - Parent process created child test1 with PID: 579.
Enter a command (create/list/terminate/create_thread/send_message/receive_messages/exit): 202
3-10-27 14:46:37,695 - Child process test1 (PID: 579) created.
create
Enter the process name: test2
2023-10-27 14:46:43,902 - Parent process created child test2 with PID: 580.
Enter a command (create/list/terminate/create_thread/send_message/receive_messages/exit): 202
3-10-27 14:46:43,902 - Child process test2 (PID: 580) created.
list
2023-10-27 14:46:45,241 - List of running processes:
2023-10-27 14:46:45,241 - PID: 579, Name: test1
2023-10-27 14:46:45,242 - PID: 580, Name: test2
Enter a command (create/list/terminate/create_thread/send_message/receive_messages/exit): sen
d_message
Enter the source PID: 579
Enter the target PID: 580
Enter the message: hello
Enter a command (create/list/terminate/create_thread/send_message/receive_messages/exit): rec
eive_message
2023-10-27 14:47:48,150 - Invalid command. Please try again.
Enter a command (create/list/terminate/create_thread/send_message/receive_messages/exit): rec
eive_messages
Enter the PID to receive messages: 580
2023-10-27 14:48:12,632 - Process test2 received message: From test1: hello
Enter a command (create/list/terminate/create_thread/send_message/receive_messages/exit):
```

❖ Process Synchronization:

Introduction:

Process synchronization is a critical aspect of multi-process programming, and it helps ensure that processes or threads work together in a coordinated and orderly manner. Common synchronization mechanisms include locks, semaphores, and condition variables.

Example Code with explanation:

```
1  import multiprocessing
2  import time
3  |
4  buffer = multiprocessing.Queue(maxsize=5)
5  mutex = multiprocessing.Lock()
6  empty = multiprocessing.Semaphore(5)
7  full = multiprocessing.Semaphore(0)
8  |
9  def producer():
10     for i in range(10):
11         empty.acquire()
12         mutex.acquire()
13         item = f"Item {i}"
14         buffer.put(item)
15         print(f"Produced: {item}")
16         mutex.release()
17         full.release()
18         time.sleep(1)
19 |
20 def consumer():
21     for i in range(10):
22         full.acquire()
23         mutex.acquire()
24         item = buffer.get()
25         print(f"Consumed: {item}")
26         mutex.release()
27         empty.release()
28         time.sleep(1)
29 |
30 if __name__ == "__main__":
31     producer_process = multiprocessing.Process(target=producer)
32     consumer_process = multiprocessing.Process(target=consumer)
33 |
34     producer_process.start()
35     consumer_process.start()
36 |
37     producer_process.join()
38     consumer_process.join()
```

Here, I have provided an example of using locks (mutex) for process synchronization within the Process Manager project. In this code, we have two processes: the producer and the consumer. The producer produces items and adds them to a shared buffer, while the consumer consumes items from the buffer. To ensure that the producer and consumer do not access the buffer simultaneously, a lock is used to synchronize access.

Complete Code:

```
1  import os
2  import sys
3  import threading
4  import time
5  import logging
6  from queue import Queue
7
8  logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(message)s')
9
10 class Process:
11     def __init__(self, name):
12         self.name = name
13         self.pid = None
14         self.threads = []
15         self.message_queue = Queue()
16         self.lock = threading.Lock() # Mutex (Lock) for process-level synchronization
17
18 class Thread:
19     def __init__(self, name):
20         self.name = name
21         self.thread = None
22
23 class ProcessManager:
24     def __init__(self):
25         self.processes = {}
26         self.lock = threading.Lock()
27
28     def create_process(self, name):
29         with self.lock:
30             pid = os.fork()
31             if pid == 0:
32                 # Child process
33                 logging.info(f"Child process {name} (PID: {os.getpid()}) created.") # Simulate process execution
34                 sys.exit()
35             else:
36                 self.processes[pid] = Process(name)
37                 self.processes[pid].pid = pid
38                 logging.info(f"Parent process created child {name} with PID: {pid}.")
39
40     def list_processes(self):
41         with self.lock:
42             logging.info("List of running processes:")
43             for pid, process in self.processes.items():
44                 logging.info(f"PID: {pid}, Name: {process.name}")
45
46     def terminate_process(self, pid):
47         with self.lock:
```

```

39
40     def list_processes(self):
41         with self.lock:
42             logging.info("List of running processes:")
43             for pid, process in self.processes.items():
44                 logging.info(f"PID: {pid}, Name: {process.name}")
45
46     def terminate_process(self, pid):
47         with self.lock:
48             if pid in self.processes:
49                 os.kill(pid, 9) # Kill the process
50                 del self.processes[pid]
51                 logging.info(f"Process with PID {pid} terminated.")
52             else:
53                 logging.error(f"Process with PID {pid} not found.")
54
55     def create_thread(self, process_pid, thread_name):
56         with self.lock:
57             if process_pid in self.processes:
58                 process = self.processes[process_pid]
59                 thread = Thread(thread_name)
60                 thread.thread = threading.Thread(target=self.thread_function, args=(thread.name,))
61                 process.threads.append(thread)
62                 thread.thread.start()
63                 logging.info(f"Thread {thread.name} created in process {process.name} (PID: {process_pid})")
64
65     def thread_function(self, thread_name):
66         logging.info(f"Thread {thread_name} started.")
67         command = input("Enter a command (create/list/terminate/create_thread/send_message/receive_message): ")
68         if command == "create":
69             name = input("Enter the process name: ")
70             self.create_process(name)
71         elif command == "list":
72             self.list_processes()
73         elif command == "terminate":
74             pid = int(input("Enter the PID of the process to terminate: "))
75             self.terminate_process(pid)
76         elif command == "create_thread":
77             pid = int(input("Enter the PID of the process to create a thread in: "))
78             thread_name = input("Enter the thread name: ")
79             self.create_thread(pid, thread_name)
80         elif command == "send_message":
81             source_pid = int(input("Enter the source PID: "))
82             target_pid = int(input("Enter the target PID: "))
83             message = input("Enter the message: ")
84             self.send_message(source_pid, target_pid, message)
85         elif command == "receive_messages":

```

```

78             thread_name = input("Enter the thread name: ")
79             self.create_thread(pid, thread_name)
80         elif command == "send_message":
81             source_pid = int(input("Enter the source PID: "))
82             target_pid = int(input("Enter the target PID: "))
83             message = input("Enter the message: ")
84             self.send_message(source_pid, target_pid, message)
85         elif command == "receive_messages":
86             pid = int(input("Enter the PID to receive messages: "))
87             self.receive_messages(pid)
88         elif command == "exit":
89             logging.info("Exiting the Process Manager.")
90             sys.exit()
91         else:
92             logging.error("Invalid command. Please try again.")
93
94         logging.info(f"Thread {thread_name} finished.")
95
96     def send_message(self, source_pid, target_pid, message):
97         if source_pid in self.processes and target_pid in self.processes:
98             source_process = self.processes[source_pid]
99             target_process = self.processes[target_pid]
100             with target_process.lock: # Mutex (Lock) for process-level synchronization
101                 target_process.message_queue.put(f"From {source_process.name}: {message}")
102         else:
103             logging.error("Invalid source or target PID.")
104
105     def receive_messages(self, pid):
106         if pid in self.processes:
107             process = self.processes[pid]
108             while not process.message_queue.empty():
109                 message = process.message_queue.get()
110                 logging.info(f"Process {process.name} received message: {message}")
111         else:
112             logging.error("Invalid PID.")
113
114 if __name__ == "__main__":
115     process_manager = ProcessManager()
116
117     while True:
118         command = input("Enter a command (create/list/terminate/create_thread/send_message/receive_message): ")
119         if command == "create":
120             name = input("Enter the process name: ")
121             process_manager.create_process(name)
122         elif command == "list":
123             process_manager.list_processes()
124         elif command == "terminate":

```



```

110         logging.info(f"Process {process.name} received message: {message}")
111     else:
112         logging.error("Invalid PID.")
113
114 if __name__ == "__main__":
115     process_manager = ProcessManager()
116
117     while True:
118         command = input("Enter a command (create/list/terminate/create_thread/send_message/receive_message): ")
119         if command == "create":
120             name = input("Enter the process name: ")
121             process_manager.create_process(name)
122         elif command == "list":
123             process_manager.list_processes()
124         elif command == "terminate":
125             pid = int(input("Enter the PID of the process to terminate: "))
126             process_manager.terminate_process(pid)
127         elif command == "create_thread":
128             pid = int(input("Enter the PID of the process to create a thread in: "))
129             thread_name = input("Enter the thread name: ")
130             process_manager.create_thread(pid, thread_name)
131         elif command == "send_message":
132             source_pid = int(input("Enter the source PID: "))
133             target_pid = int(input("Enter the target PID: "))
134             message = input("Enter the message: ")
135             process_manager.send_message(source_pid, target_pid, message)
136         elif command == "receive_messages":
137             pid = int(input("Enter the PID to receive messages: "))
138             process_manager.receive_messages(pid)
139         elif command == "exit":
140             logging.info("Exiting the Process Manager.")
141             sys.exit()
142         else:
143             logging.error("Invalid command. Please try again.")
144

```

Output:

```

mitpatel101@Mits-Spectre:/mnt/c/Users/patel/OneDrive - The Pennsylvania State University/2023
/Fall/CMPSC 472/Project 1$ python3 Process-Manager.py
Enter a command (create/list/terminate/create_thread/send_message/receive_messages/exit): cre
ate
Enter the process name: test1
2023-10-27 14:55:47,268 - Parent process created child test1 with PID: 649.
2023-10-27 14:55:47,269 - Child process test1 (PID: 649) created.
Enter a command (create/list/terminate/create_thread/send_message/receive_messages/exit): cre
ate
Enter the process name: test2
2023-10-27 14:55:51,258 - Parent process created child test2 with PID: 650.
Enter a command (create/list/terminate/create_thread/send_message/receive_messages/exit): 202
3-10-27 14:55:51,259 - Child process test2 (PID: 650) created.
create
Enter the process name: test3
2023-10-27 14:55:54,755 - Parent process created child test3 with PID: 651.
Enter a command (create/list/terminate/create_thread/send_message/receive_messages/exit): 202
3-10-27 14:55:54,756 - Child process test3 (PID: 651) created.
list
2023-10-27 14:55:57,128 - List of running processes:
2023-10-27 14:55:57,128 - PID: 649, Name: test1
2023-10-27 14:55:57,128 - PID: 650, Name: test2
2023-10-27 14:55:57,128 - PID: 651, Name: test3
Enter a command (create/list/terminate/create_thread/send_message/receive_messages/exit): ter
minate
Enter the PID of the process to terminate: 649
2023-10-27 14:56:02,822 - Process with PID 649 terminated.
Enter a command (create/list/terminate/create_thread/send_message/receive_messages/exit): cre
ate_thread
Enter the PID of the process to create a thread in: 650
Enter the thread name: thread1
2023-10-27 14:56:23,428 - Thread thread1 started.
Enter a command (create/list/terminate/create_thread/send_message/receive_messages/exit): 202
3-10-27 14:56:23,428 - Thread thread1 created in process test2 (PID: 650).

```

```

list
Enter a command (create/list/terminate/create_thread/send_message/receive_messages/exit): 202
3-10-27 14:57:35,844 - List of running processes:
2023-10-27 14:57:35,844 - PID: 650, Name: test2
2023-10-27 14:57:35,844 - PID: 651, Name: test3
2023-10-27 14:57:35,844 - Thread thread1 finished.
send_message
Enter the source PID: 650
Enter the target PID: 651
Enter the message: hello
Enter a command (create/list/terminate/create_thread/send_message/receive_messages/exit): rec
eive_messages
Enter the PID to receive messages: 651
2023-10-27 14:57:59,222 - Process test3 received message: From test2: hello
Enter a command (create/list/terminate/create_thread/send_message/receive_messages/exit): exi
t
2023-10-27 14:58:03,682 - Exiting the Process Manager.

```

Output Explanation:

- After running the process manager, I created 3 processes: test1, test2 and test3 to show the functionality of the creating process. The output of each command is well organized and easy to understand and gives logs and reports on each command execution.
- After that I list all the processes using the “list” command and then terminate the test3 process using the terminate command.
- Then, I create the thread using the “create_thread” command and name it as; thread1. After creating the thread1, again a menu of processes manager appears which shows that the thread is running as a separate process. In the thread function, I have provided the function to use the functionalities of the process manager so that we should have access to process manager functionalities in this separate thread. We used to create functionality to create process thread1process and we see in the output that new process thread1process is created and then thread1 finished.
- Then, I tested the IPC functionality by sending a message “hello” from process test1 to test2 by using send_messages and receiving the message by process test2 using receive_messages.
- The “exit” command closes the program.