  
  
  
CA1 - Linux Environment

Programming Assignment

Student Name: Gábor Major

Student Number: C00271548

Date Due: 26/10/2023 Lecturer Name: Caroline Byrne

Table of Contents

[Introduction 2](#__RefHeading___Toc291_100016948)

[Process Management 3](#__RefHeading___Toc293_100016948)

[Process Creation 4](#__RefHeading___Toc304_100016948)

[Process Scheduling 5](#__RefHeading___Toc306_100016948)

[Process Destruction 6](#__RefHeading___Toc308_100016948)

[Conclusion 7](#__RefHeading___Toc295_100016948)

[Bibliography 7](#__RefHeading___Toc297_100016948)

[Declaration Form 8](#__RefHeading___Toc884_2771911116)

[Collaboration Form 8](#__RefHeading___Toc886_2771911116)

[Appendix 9](#__RefHeading___Toc681_1510207793)

Word Count: 1,519

# Introduction

In this document Linux processes will be discussed, including process management, process creation, process scheduling, and process destruction.

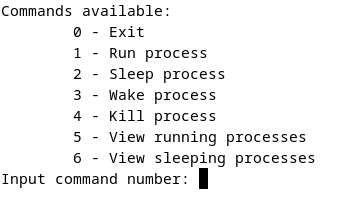
Processes are active entities whereas programs are static entities. For this reason there is more than just the program code present in a process. This data is stored in the process control block. It requires the following attributes to run properly and be managed by the operating system:

* **Process Id:** Unique number of the process.
* **Process Sate:** State in which the process is in.
* **Central Processing Unit Registers:** Stores the details of the registers when swapped out.
* **Accounts Information:** Holds information about execution time, start time, time limits.
* **Input Output Information:** Holds information about devices or files used.
* **Central Processing Unit Scheduling Information:** Stores the priority of the process.

(GeeksforGeeks, 2023).

Along with this document there is also a Python process management program which demonstrates how process management works on a high level. It is a command line interface program which can; create processes, sleep processes, wake processes, kill processes, view all of the running and sleeping processes created by the program. To use the program you must input the command number and then input information related to the command for example the name of the program you want to run.

For security and system integrity reasons the Python program is able to create any process, but is unable to schedule or terminate any other processes not created by the program itself. This stops the program from accidentally causing system malfunction.

**Figure 1:** Python program interface. **Source:** Author, 2023.

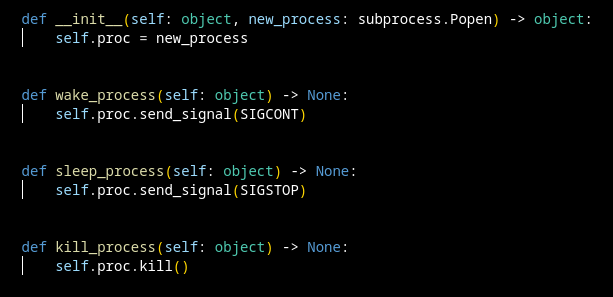
# Process Management

Process management is part of the job of the operating system, which includes jobs related to processes like creating processes, scheduling processes and destroying processes. Process management is when the operating system specifies system resources like memory amount, central processing unit usage, and graphical processing unit usage.  
The operating system is also in charge of making sure the processes only access their own resources, keeping the system secure. (M. Sagar, 2023).

Processes can be in different many states during their lifetime:

* **New:** Process being created.
* **Ready:** Process ready to be executed.
* **Running:** Process is being run.
* **Waiting:** Process is waiting for an event to happen either waiting for I/O events or other processes.
* **Terminated:** Process has finished execution.

Switching between these states and processes happens all the time and is called context switching. It takes time to transition from one process to another and if this happens a lot it is said that the operating system is thrashing. To avoid this process management is of crucial importance to keep operations smooth. Context switching can happen for the following reasons:

* **Time Slicing:** This happens when the time allocated for a process is used up and the process is swapped out for a different one to give resources over.
* **Interrupt Handling:** This happens when a type of interrupt happens either a hardware interrupt or a software interrupt. A context switch then happens for the central processing unit to handle the interrupt like a keyboard input or system call, then resumes the previous process.
* **Process State Change:** This happens when a process changes state, for example from waiting to ready or vice versa.
* **Blocking Operations:** This happens when a process calls an operation that requires an input or output, from or to an external device, for example input from keyboard, or writing to disk.

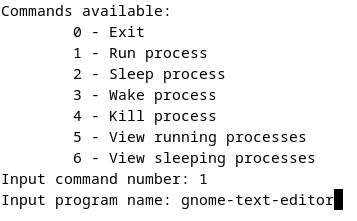
(M. Sagar, 2023).

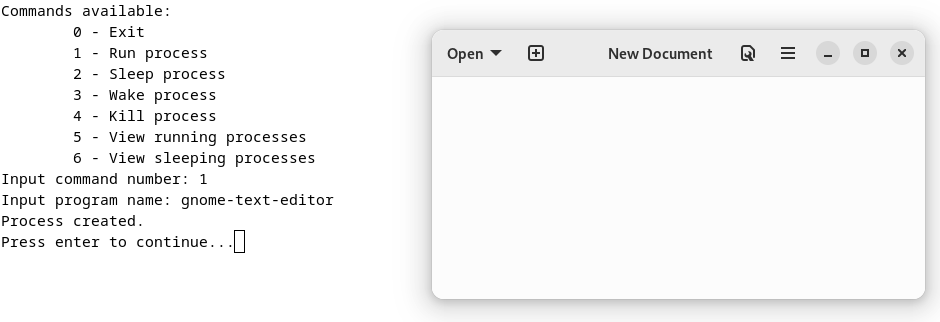
In the Python program all of the communication with the system process takes place in a custom Python class.

# Process Creation

Process are created by the operating system. They are created from a program which is passive code stored on a storage medium like a hard drive or solid state drive, and turned into a process which lives in system memory and is code that is active.  
Even though the program is stored once, it may open many different processes for different users or even create new processes itself. (GeeksforGeeks, 2023).

When a process is first started from a program it is put into the New state, in this state it cannot run until it has been correctly loaded into memory.  
After that the process is changed into the Ready state, the process is now correctly loaded into system memory, and is ready to be executed.  
The following figures show the Python process management program creating a new process for a text editor.

**Figure 2:** Program starting text editor. **Source:** Author, 2023.

**Figure 3:** Program started new process, which is now ready. **Source:** Author, 2023.

# Process Scheduling

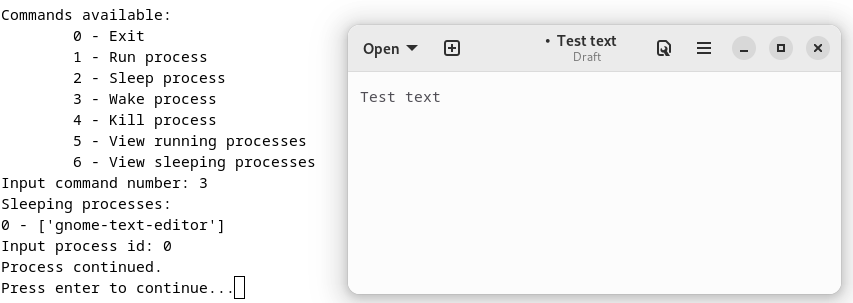
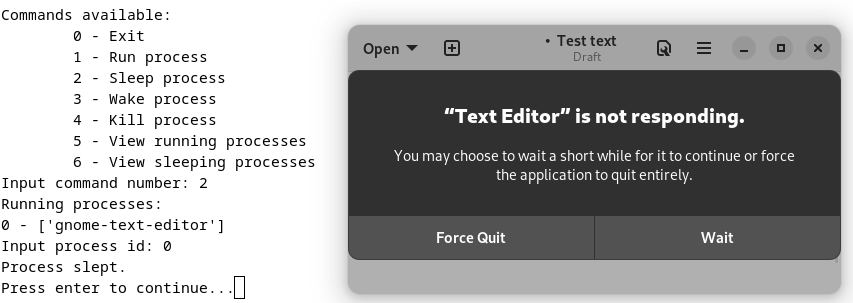
Operating systems also have the role of managing all of the currently ready processes. If all of the processes could choose when they get to execute it would be mayhem as all developers would want their own code to run the fastest and have access to the most amount of resources.  
This is the reason that processes need to be scheduled so each process can run properly. The operating system keeps track of which processes are running, what they are using, and when the process was last executed. (M. Sagar, 2023).

To fix the issue of maintaining processes there exists many scheduling algorithms:

* **First-come, first-served:** This is the easiest algorithm to implement as only a process queue is required. When a process is in the Ready state it gets added to the end of the queue, and processes are removed from the front of the queue when the central processing unit is ready.
* **Shortest Job First:** This algorithm looks at data about the process before it runs it. It looks at how long the process is estimated to be run for and always runs the process in the ready list with the shortest time.
* **Round Robin:** Round Robin is a different algorithm again which focuses more on giving each process an equal chance of running. It achieves this by having a set amount of time, for which each process is allowed to run for. If the process does not finish in the set amount of time it is put in the Ready state and put in the back of the queue.
* **Priority Scheduling:** Finally this algorithm takes into consideration what type each process is and runs them based on that. This algorithm can change on what the user is doing, for example, watching a video, editing documents, or playing games.

(GeeksforGeeks, 2023).

The Python program allows for the user to manually set which processes are awake and running and which ones are sleeping, these processes cannot run unless explicitly they get permission.

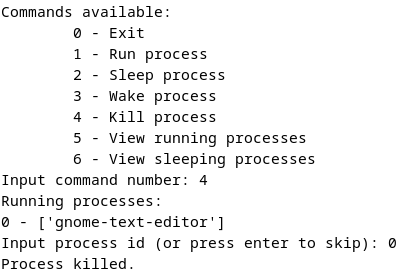
**Figure 4:** Text editor slept, and running. **Source:** Author, 2023.

# Process Destruction

Terminating or ending processes is done by the operating system as well, this can happen manually when the user wants to or when the operating system is done with a service. The operating system goes through a set routine to make sure all of resources that were used by the process are freed up for the system. The operating system tells the process to close, and stop all operations then removes the process from the process table. It is important for the system to properly deallocate the resources so the system keeps functioning. (M. Sagar, 2023).

When the Python process management program is run and the kill process option is selected, it shows a list of process currently running and sleeping that were started by it, it then needs to be given an id from the list so it knows which process to end. The program is only able to terminate processes it itself created as to not cause the user to terminate a critical system processes.  
The program also checks if any process is still active on exit, either running or sleeping and asks the user whether they want to terminate them, or to keep them running in the background after the Python program has exited.

The data that was stored about the process that the Python program started, is then taken out of either the running list or sleeping list and all of the memory is deallocated for the rest of the operating system and so the Python program does not become bloated and have memory leak issues.

**Figure 5:** Program successfully stopped process. **Source:** Author, 2023.

Process termination is a very important job of the operating system, it is of crucial importance that the operating system frees up memory, execution slots and closes files to keep functioning correctly.

# Conclusion

This report went through a high level understanding of how operating systems manage different processes, including process creation, process scheduling, and process termination. If the operating system would not do any process management, all the processes would be in mayhem and may cause unexpected errors.

Process management is a crucial and important part of computing, especially of multi process and multi threaded system, where many processes compete for resources. It allows for the operating system to process multiple tasks at the same time, and run fancy graphical user interfaces and animations. This however has the disadvantage of requiring more resources like memory and storage and adding complexity to the operating system.

Along with the report there Is a Python program which shows have process management can work if manually done by the user. This is less efficient than algorithms used by operating systems which may use different ones depending on the situation.

# Bibliography

References for report:

(GeeksforGeeks, 2023) GeeksforGeeks (2023) Introduction of Process Management. Available at: <https://www.geeksforgeeks.org/introduction-of-process->management/ (Accessed: 25 September 2023).

(M. Sagar, 2023) Sagar, M. (2023) *Introduction of Process Management.* Available at: <https://www.codingninjas.com/studio/library/introduction-to-process->management (Accessed: 25 September 2023).

References for Python program:

Python 3.11.5 documentation (2023) *subprocess — Subprocess management.* Available at:<https://docs.python.org/3/library/subprocess.html> (Accessed: 20 September 2023).

Kerrisk, M. (2023) *signal(7) — Linux manual page.* Available at:

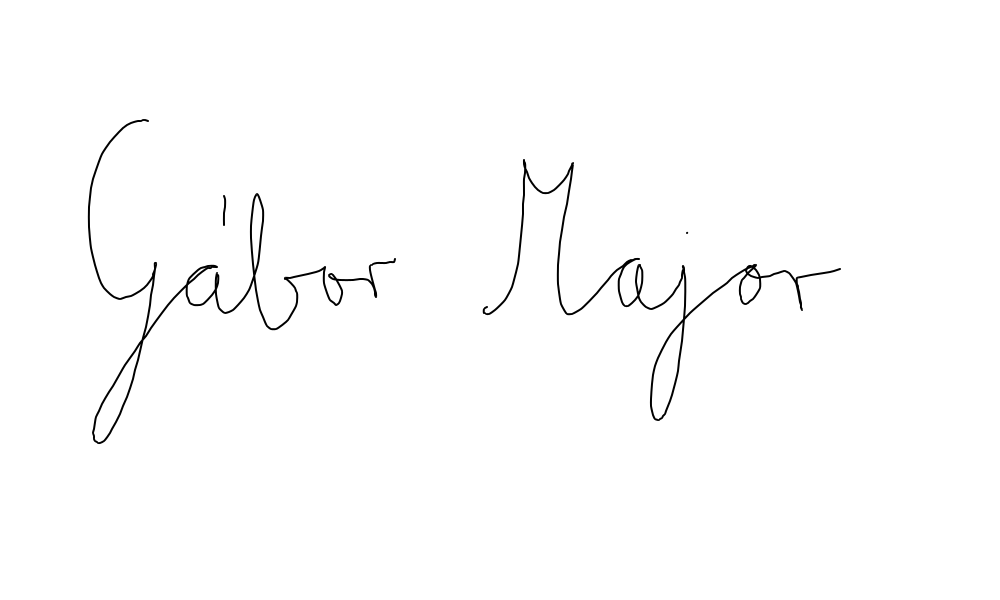
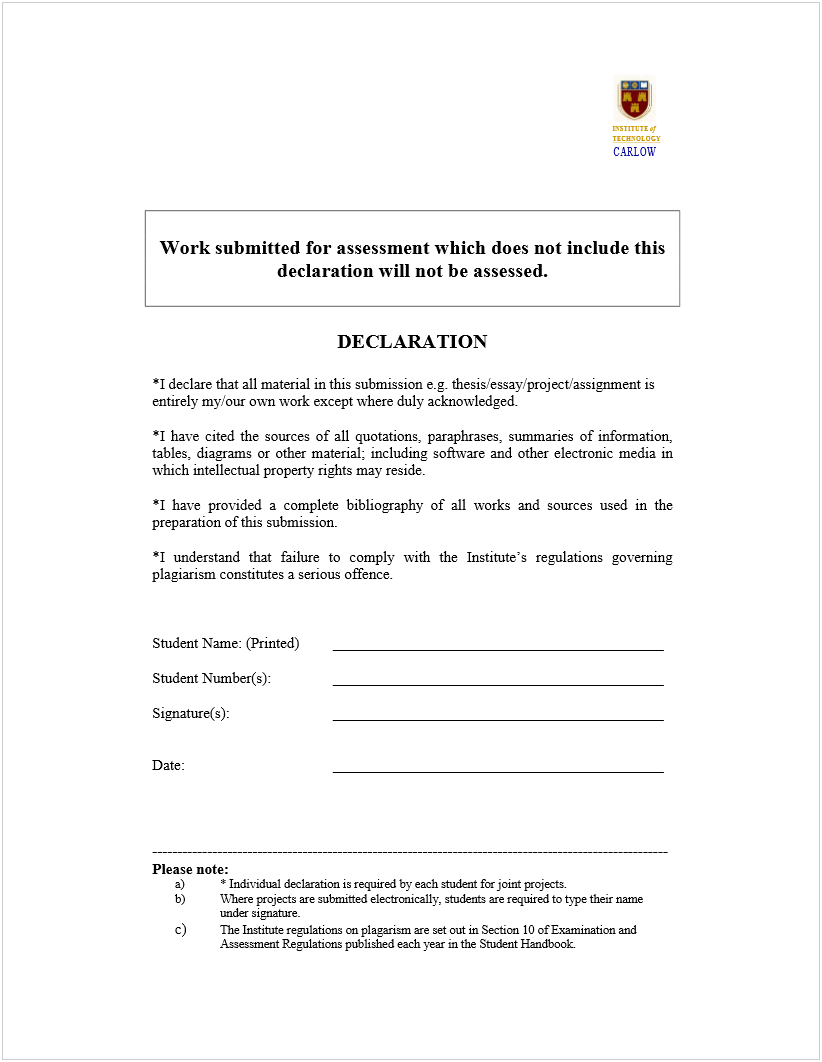
<https://www.man7.org/linux/man-pages/man7/signal.7.html> (Accessed: 20 September 2023).

# Declaration Form

Gábor Major

C00271548

12.10.2023



# Collaboration Form

**Student Name:\_\_\_\_\_\_\_\_Gábor Major\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Student Number:\_\_\_\_\_\_C00271548\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**I have not collaborated with anybody whilst conducting research for this assignment.**

# Appendix

Full Python Program Code:

1. import subprocess
2. import time
3. ## signal for waking process
4. SIGCONT: int = 18
5. ## signal for sleeping process
6. SIGSTOP: int = 19
7. class ProcessClass:
8. """
9. class for process
10. stores Popen class
11. """
12. proc: subprocess.Popen = None
13. def \_\_init\_\_(self: object, new\_process: subprocess.Popen) -> object:
14. self.proc = new\_process
15. def wake\_process(self: object) -> None:
16. self.proc.send\_signal(SIGCONT)
17. def sleep\_process(self: object) -> None:
18. self.proc.send\_signal(SIGSTOP)
19. def kill\_process(self: object) -> None:
20. self.proc.kill()
21. ## prints out the commands
22. def print\_commands() -> None:
23. info: str = """Commands available:
24. 0 - Exit
25. 1 - Run process
26. 2 - Sleep process
27. 3 - Wake process
28. 4 - Kill process
29. 5 - View running processes
30. 6 - View sleeping processes"""
31. print(info)
32. return
33. ## asks to stop processes and then quits
34. def exit\_program(process\_dictionary: dict) -> None:
35. if len(process\_dictionary["Running"]) != 0:
36. view\_running\_list(process\_dictionary)
37. confirmation: str = input("There are running processes, would you ike to kill them? (y / n): ")
38. if confirmation == "y":
39. for process in process\_dictionary["Running"]:
40. process.kill\_process()
41. process\_dictionary["Running"] = []
42. if len(process\_dictionary["Sleeping"]) != 0:
43. view\_sleeping\_list(process\_dictionary)
44. confirmation: str = input("There are sleeping processes, would you ike to kill them? (y / n): ")
45. if confirmation == "y":
46. for process in process\_dictionary["Sleeping"]:
47. process.kill\_process()
48. process\_dictionary["Sleeping"] = []
49. confirmation: str = input("Are you sure you want to exit? (y / n): ")
50. if confirmation == "y":
51. quit()
52. return
53. ## creates new Popen and ProcessClass process
54. ## adds the new ProcessClass to the active list
55. def create\_new\_process(process\_dictionary: dict) -> None:
56. program\_name: str = input("Input program name: ")
57. ## checks if the program exists
58. try:
59. new\_process: subprocess.Popen = subprocess.Popen([program\_name])
60. except FileNotFoundError:
61. print("Program not found!")
62. return
63. process\_dictionary["Running"].append(ProcessClass(new\_process))
64. print("Process created.")
65. return
66. ## sleeps a process and moves them to the sleeping list
67. def sleep\_process(process\_dictionary: dict) -> None:
68. view\_running\_list(process\_dictionary)
69. process\_index: int = input("Input process id: ")
70. try:
71. process\_index = int(process\_index)
72. except:
73. print("Not number input!")
74. return
75. if process\_index < len(process\_dictionary["Running"]):
76. process: ProcessClass = process\_dictionary["Running"][abs(process\_index)]
77. process\_dictionary["Running"].remove(process)
78. process.sleep\_process()
79. process\_dictionary["Sleeping"].append(process)
80. print("Process slept.")
81. else:
82. print("Id not found!")
83. return
84. ## sleeps a process and moves them to the running list
85. def wake\_process(process\_dictionary: dict) -> None:
86. view\_sleeping\_list(process\_dictionary)
87. process\_index: int = input("Input process id: ")
88. try:
89. process\_index = int(process\_index)
90. except:
91. print("Not number input!")
92. return
93. if process\_index < len(process\_dictionary["Sleeping"]):
94. process: ProcessClass = process\_dictionary["Sleeping"][abs(process\_index)]
95. process\_dictionary["Sleeping"].remove(process)
96. process.wake\_process()
97. process\_dictionary["Running"].append(process)
98. print("Process continued.")
99. else:
100. print("Id not found!")
101. return
102. ## kills a process and removes it from its respective list
103. def kill\_process(process\_dictionary: dict) -> None:
104. view\_running\_list(process\_dictionary)
105. process\_index: int = input("Input process id (or press enter to skip): ")
106. try:
107. process\_index = int(process\_index)
108. except:
109. if process\_index != "\n":
110. print("Not number input!")
111. process\_index = "\n"
112. if process\_index == "\n":
113. pass
114. elif process\_index < len(process\_dictionary["Running"]):
115. process: ProcessClass = process\_dictionary["Running"][abs(process\_index)]
116. process\_dictionary["Running"].remove(process)
117. process.kill\_process()
118. print("Process killed.")
119. else:
120. print("Id not found!")
121. view\_sleeping\_list(process\_dictionary)
122. process\_index: int = input("Input process id (or press enter to skip): ")
123. try:
124. process\_index = int(process\_index)
125. except:
126. if process\_index != "\n":
127. print("Not number input!")
128. process\_index = "\n"
129. if process\_index == "\n":
130. pass
131. elif process\_index < len(process\_dictionary["Sleeping"]):
132. process: ProcessClass = process\_dictionary["Sleeping"][abs(process\_index)]
133. process\_dictionary["Sleeping"].remove(process)
134. process.kill\_process()
135. print("Process killed.")
136. else:
137. print("Id not found!")
138. return
139. ## print out the running processes
140. def view\_running\_list(process\_dictionary: dict) -> None:
141. print("Running processes:")
142. if len(process\_dictionary["Running"]) == 0:
143. print("None")
144. else:
145. for index in range(len(process\_dictionary["Running"])):
146. print(f"{index} - {process\_dictionary['Running'][index].proc.args}")
147. return
148. ## print out the sleeping processes
149. def view\_sleeping\_list(process\_dictionary: dict) -> None:
150. print("Sleeping processes:")
151. if len(process\_dictionary["Sleeping"]) == 0:
152. print("None")
153. else:
154. for index in range(len(process\_dictionary["Sleeping"])):
155. print(f"{index} - {process\_dictionary['Sleeping'][index].proc.args}")
156. return
157. def main() -> None:
158. process\_dictionary = {"Running": [],
159. "Sleeping": []}
160. while True:
161. print\_commands()
162. input\_command: int = input("Input command number: ")
163. try:
164. input\_command = int(input\_command)
165. except:
166. print("Not number input!")
167. continue
168. if input\_command == 0:
169. exit\_program(process\_dictionary)
170. elif input\_command == 1:
171. create\_new\_process(process\_dictionary)
172. elif input\_command == 2:
173. sleep\_process(process\_dictionary)
174. elif input\_command == 3:
175. wake\_process(process\_dictionary)
176. elif input\_command == 4:
177. kill\_process(process\_dictionary)
178. elif input\_command == 5:
179. view\_running\_list(process\_dictionary)
180. elif input\_command == 6:
181. view\_sleeping\_list(process\_dictionary)
182. else:
183. print("Command not found!")
184. input("Press enter to continue...")
185. return
186. if \_\_name\_\_ == "\_\_main\_\_":
187. main()