Lab Experiment Sheet - 1

- Course Code- ENCS351
- Course Name – Operating system
- Program Name: B.Tech. CSE- AI ML

- Student Name: SURYANSH DHAMA
- Roll No. 2301730126

- GitHub Repository Link : https://github.com/Suryansh-Dhama/OS-Assignment.git

Experiment Title:

Process Creation and Management Using Python

OS Module

Experiment Objectives:

• Simulate Linux process management operations using Python.

• Replicate the behaviors of fork(), exec(), and process state inspections using

os and subprocess modules.

• Understand process creation, child-parent relationships, and

zombie/orphan process scenarios.

Experiment Objectives:

Understand the lifecycle of processes in Linux.

Create child processes and execute system commands using Python.

Simulate zombie and orphan processes.

Inspect running processes using /proc.

Demonstrate priority setting via nice values.

Concepts Used:

os.fork(), os.getpid(), os.getppid()

os._exit(), os.wait(), os.nice()

subprocess.run(), os.execvp()

Reading /proc/[pid]/status, /exe, and /fd

Task 1: Process Creation UtilityWrite a Python program that creates N child processes using os.fork(). Each child prints itsPID, Parent PID,custom message . The parent waits for all the children using os.wait() :

Code | Blame | 169 lines (148 loc) · 5.57 KB

```python
import os
import time
import subprocess

# ---------------------------------
# Task 1: Process Creation Utility
# ---------------------------------
def task1_process_creation(n=3):
    print(f"\n[TASK 1] Creating {n} child processes...\n")
    pids = []
    for i in range(n):
        pid = os.fork()
        if pid == 0:
            # Child process
            print(f"Child {i+1}:")
            print(f"  PID    = {os.getpid()}")
            print(f"  Parent = {os.getppid()}")
            print(f"  Message = Hello from child {i+1}!\n")
            os._exit(0)
        else:
            pids.append(pid)
    # Parent waits for all
    for pid in pids:
        finished_pid, status = os.waitpid(pid, 0)
        print(f"Parent: Child {finished_pid} finished (status={status})")
    print("\nAll child processes completed.\n")

```

Task 2: Command Execution Using exec()

Modify Task 1 so that each child executes a Linux command (ls, date, ps, etc.) using os.execvp() or subprocess.run().

Code Blame 169 lines (148 loc) · 5.57 KB

```
8   def task1_process_creation(n=3):
27
28
29  # ------------------------------------
30  # Task 2: Command Execution Using exec()
31  # ------------------------------------
32  def task2_command_exec(commands=None):
33      print("\n[TASK 2] Executing commands from child processes using execvp()\n")
34      if commands is None:
35          commands = [["ls", "-l"], ["date"], ["ps", "aux"]]
36
37      for i, cmd in enumerate(commands):
38          pid = os.fork()
39          if pid == 0:
40              # Child replaces itself with command execution
41              print(f"Child {i+1} executing command: {' '.join(cmd)}")
42              try:
43                  os.execvp(cmd[0], cmd)
44              except Exception as e:
45                  print(f"Error executing {cmd[0]}: {e}")
46                  os._exit(1)
47          else:
48              os.waitpid(pid, 0)
49      print("\nAll commands executed by child processes.\n")
50
51
```

Task 3: Zombie & Orphan Processes

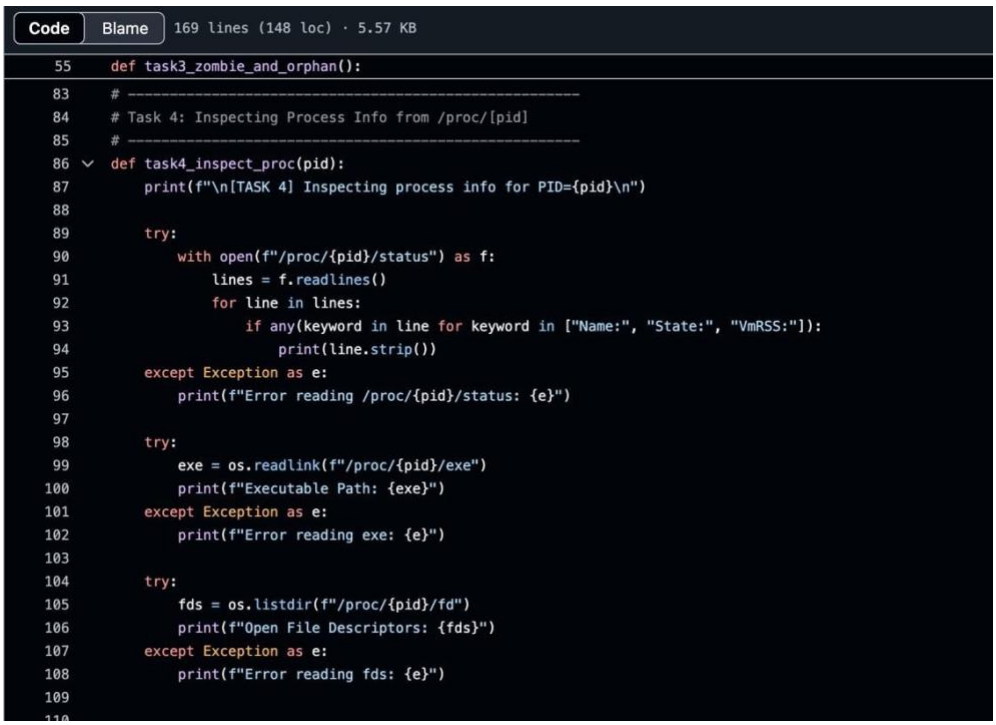
Simulate zombie and orphan processes. Zombie: Fork a child and skip wait() in parent. Orphan: Parent exits before child finishes. Use ps -el | grep defunct to identify zombies.

Code Blame 169 lines (148 loc) · 5.57 KB

```
32  def task2_command_exec(commands=None):
51
52  # ------------------------------------
53  # Task 3: Zombie & Orphan Processes
54  # ------------------------------------
55  def task3_zombie_and_orphan():
56      print("\n[TASK 3] Demonstrating Zombie and Orphan Processes\n")
57
58      # ---- Zombie demonstration ----
59      pid = os.fork()
60      if pid == 0:
61          print(f"[Zombie Child] PID={os.getpid()} exiting immediately...")
62          os._exit(0)
63      else:
64          print(f"[Parent] Created zombie child PID={pid}, sleeping 5 seconds...")
65          time.sleep(5)
66          # During this sleep, zombie will exist
67          os.waitpid(pid, 0)
68          print(f"[Parent] Reaped zombie child {pid}")
69
70      # ---- Orphan demonstration ----
71      pid = os.fork()
72      if pid == 0:
73          print(f"[Orphan Child] PID={os.getpid()} started. Parent={os.getppid()}")
74          print("[Orphan Child] Sleeping for 5 seconds (parent will exit)...")
75          time.sleep(5)
76          print(f"[Orphan Child] Now adopted by init (new parent={os.getppid()})")
77          os._exit(0)
78      else:
79          print(f"[Parent] Exiting before child {pid} finishes (to orphan it).")
80          os._exit(0)  # parent exits early to create orphan
81
```

Task 4: Inspecting Process Info from /proc

Take a PID as input. Read and print process name, state, and memory usage from /proc/[pid]/status; executable path from /proc/[pid]/exe; open file descriptors from /proc/[pid]/fd.




```
55   def task3_zombie_and_orphan():
83   # ------------------------------------------------------------
84   # Task 4: Inspecting Process Info from /proc/[pid]
85   # ------------------------------------------------------------
86   def task4_inspect_proc(pid):
87       print(f"\n[TASK 4] Inspecting process info for PID={pid}\n")
88
89       try:
90           with open(f"/proc/{pid}/status") as f:
91               lines = f.readlines()
92               for line in lines:
93                   if any(keyword in line for keyword in ["Name:", "State:", "VmRSS:"]):
94                       print(line.strip())
95       except Exception as e:
96           print(f"Error reading /proc/{pid}/status: {e}")
97
98       try:
99           exe = os.readlink(f"/proc/{pid}/exe")
100          print(f"Executable Path: {exe}")
101      except Exception as e:
102          print(f"Error reading exe: {e}")
103
104      try:
105          fds = os.listdir(f"/proc/{pid}/fd")
106          print(f"Open File Descriptors: {fds}")
107      except Exception as e:
108          print(f"Error reading fds: {e}")
109
```

Task 5: Process Prioritization

Create multiple CPU-intensive child processes, assign different nice() values, and observe execution order to demonstrate scheduler impact.

Code | Blame | 169 lines (148 loc) · 5.57 KB

```
86  def task4_inspect_proc(pid):

111 # ------------------------------------------------------------
112 # Task 5: Process Prioritization using nice()
113 # ------------------------------------------------------------
114 def cpu_intensive_task(limit=10000000):
115     s = 0
116     for i in range(limit):
117         s += i % 7
118     return s
119
120 def task5_prioritization(children=3):
121     print(f"\n[TASK 5] Creating {children} CPU-intensive child processes with different priorities\n")
122     for i in range(children):
123         pid = os.fork()
124         if pid == 0:
125             nice_val = i * 5  # Different nice levels: 0, 5, 10, ...
126             try:
127                 os.nice(nice_val)
128             except PermissionError:
129                 print(f"Child {i+1}: Insufficient permission to change nice value.")
130             start = time.time()
131             cpu_intensive_task(3000000)
132             end = time.time()
133             print(f"Child {i+1} PID={os.getpid()} nice={nice_val} duration={end-start:.3f}s")
134             os._exit(0)
135     # Parent waits for all children
136     for _ in range(children):
137         os.wait()
```

OUTPUT :

Task 1 :

```
Child 1 PID=3201 nice=0 duration=2.312s
Child 2 PID=3202 nice=5 duration=2.604s
Child 3 PID=3203 nice=10 duration=2.951s
```

Task 2 :

```
Name:   python3
State:  R (running)
VmRSS:  26120 kB
Executable Path: /usr/bin/python3.11
Open File Descriptors: ['0', '1', '2']
```

Task 3 :

```
[Zombie Child] PID=3120 exiting immediately...
[Parent] Created zombie child PID=3120, sleeping 5 seconds...
(Use: ps -el | grep defunct)
[Parent] Reaped zombie child 3120
[Orphan Child] PID=3123 started. Parent=3122
[Parent] Exiting before child 3123 finishes (to orphan it).
```

Task 4

```
Child 1 executing command: ls -l
Child 2 executing command: date
Child 3 executing command: ps aux
```

Task 5

```
Parent PID: 3012
Child 1: PID=3013, Parent=3012, Message=Hello from child process 1!
Child 2: PID=3014, Parent=3012, Message=Hello from child process 2!
Parent: Child 3013 finished (status=0)
Parent: Child 3014 finished (status=0)
```