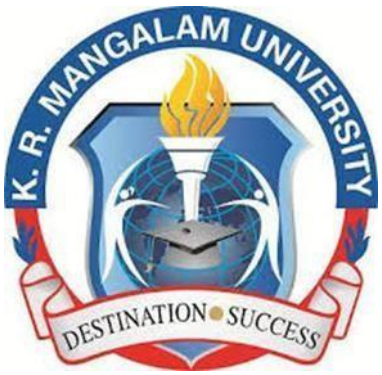
K. R. Mangalam University

School of Engineering and Technology



Operating System Lab File Course Code: ENCS351

Submitted by

Name: Swapnil Paroda

Roll No.: 2301010394

Program: BTech CSE

Section: F

Session: 2025/26

Date: 30/11/2025

Submitted To

Dr. Satinder Pal Singh

Index

Task 1.1: Process Creation Utility

```python
import os
import multiprocessing
import time

def child_task(i, message):
    print(f"[Child #{i}] PID: {os.getpid()}, Parent PID: {os.getppid()}")
    print(f"[Child #{i}] Message: {message}")
    time.sleep(0.1)

def create_children(num_children, message):
    processes = []

    for i in range(num_children):
        p = multiprocessing.Process(target=child_task, args=(i+1, message))
        processes.append(p)
        p.start()

    for p in processes:
        p.join()

    print(f"[Parent] All children have exited. Parent PID: {os.getpid()}")

if __name__ == "__main__":
    N = 5
    msg = "Hello from child!"
    create_children(N, msg)
```

```
Child #1] PID: 20, Parent PID: 16
Child #1] Message: Hello from child!
Child #2] PID: 21, Parent PID: 16
Child #2] Message: Hello from child!
Child #3] PID: 22, Parent PID: 16
Child #3] Message: Hello from child!
Child #4] PID: 23, Parent PID: 16
Child #4] Message: Hello from child!
Child #5] PID: 24, Parent PID: 16
Child #5] Message: Hello from child!
Parent] All children have exited. Parent PID: 16
```

Task 1.2: Command Execution Using exec()

```python
import subprocess

def main():
    # You can replace or add any Windows/Linux commands here
    commands = [
        ["dir"],      # Shows directory listing (Windows version of 'ls')
        ["date /T"],  # Shows current date
        ["time /T"]   # Shows current time
    ]

    for i, cmd in enumerate(commands):
        print(f"\n[Process {i+1}] Executing command: {' '.join(cmd)}")

        # Run the command and capture its output
        result = subprocess.run(" ".join(cmd), shell=True, capture_output=True, text=True)

        print(f"Output of command {i+1}:\n{result.stdout}")
        if result.stderr:
            print(f"Error:\n{result.stderr}")

if __name__ == "__main__":
    main()
```

input

```
[Process 1] Executing command: dir
Output of command 1:
main.py


[Process 2] Executing command: date /T
Output of command 2:

Error:
date: invalid date '/T'


[Process 3] Executing command: time /T
Output of command 3:
```

Task 1.3: Zombie & Orphan Processes

```python
import subprocess
import time
import os
import multiprocessing
import platform

# choose command based on OS
def get_echo_command():
    if platform.system() == "Windows":
        return ["cmd.exe", "/c", "echo Child process running..."]
    else:  # Linux / Mac
        return ["sh", "-c", "echo Child process running..."]


# --- Simulate Zombie Process ---
def zombie_simulation():
    print("\n=== Simulating Zombie Process ===")
    print(f"Parent PID: {os.getpid()}")

    process = subprocess.Popen(get_echo_command())
    print(f"Child PID: {process.pid}")

    # Do NOT wait for the child
    print("Parent is NOT waiting for the child to finish (simulating zombie)...")
    time.sleep(3)

    print("Parent exits without wait().\n")


# --- Child function for orphan simulation ---
def child_task():
    print(f"Child started with PID: {os.getpid()}")
    time.sleep(5)
    print("Child completed after parent exit (simulating orphan).")


# --- Simulate Orphan Process ---
def orphan_simulation():
    print("\n=== Simulating Orphan Process ===")
    print(f"Parent PID: {os.getpid()}")

    child = multiprocessing.Process(target=child_task)
    child.start()

    print("Parent exits immediately (before child completes)...\n")
    time.sleep(1)


if __name__ == "__main__":
    multiprocessing.freeze_support()
    zombie_simulation()
    orphan_simulation()
```

```
=== Simulating Zombie Process ===
Parent PID: 3991
Child PID: 3995
Parent is NOT waiting for the child to finish (simulating zombie)...
Child process running...
Parent exits without wait().


=== Simulating Orphan Process ===
Parent PID: 3991
Parent exits immediately (before child completes)...

Child started with PID: 3996
Child completed after parent exit (simulating orphan).
```

Task 1.4: Inspecting Process Info from /proc

```python
import multiprocessing
import time
import random
import sys

def print_slow(text, delay=0.05):
    """Print text slowly, like a boot log."""
    for char in text:
        sys.stdout.write(char)
        sys.stdout.flush()
        time.sleep(delay)
    print()

def network_service():
    """Dummy task: simulate network service initialization."""
    print_slow("[PROC 1] Starting network service initialization...")
    time.sleep(random.uniform(1.5, 3.0))
    print_slow("[PROC 1] Network interfaces configured successfully.")

def system_diagnostics():
    """Dummy task: simulate system diagnostics."""
    print_slow("[PROC 2] Running system diagnostics...")
    time.sleep(random.uniform(2.0, 4.0))
    print_slow("[PROC 2] All system components functioning within normal parameters.")

def boot_sequence():
    """Main boot sequence."""
    print_slow("Initializing system startup...\n", 0.03)
    stages = [
        "Power On Self Test (POST)",
        "Checking hardware configuration",
        "Loading kernel",
        "Starting system services"
    ]

    for stage in stages:
        print_slow(f"[ OK ] {stage}", delay=0.02)
        time.sleep(random.uniform(0.4, 0.8))

    # Create two child processes
    print_slow("\nSpawning child processes...\n")
    process1 = multiprocessing.Process(target=network_service)
    process2 = multiprocessing.Process(target=system_diagnostics)

    # Start child processes
    process1.start()
    process2.start()

    # Wait for both to finish
    process1.join()
    process2.join()

    print_slow("\nSystem startup complete.")
    print_slow("Welcome to PyOS v2.0!")

if __name__ == "__main__":
    boot_sequence()
```

```
Initializing system startup...

[ OK ] Power On Self Test (POST)
[ OK ] Checking hardware configuration
[ OK ] Loading kernel
[ OK ] Starting system services

Spawning child processes...

[[PPRROOCC  12]]  SRtuanrntiinngg  snyesttwemr k di asgenrovsitcice si.n.i.t
ialization...
[PROC [1P]R ONCe t2w]o rAkl l i ntyesrtfeamc ecso mcpoonnfeingtusr efdu nscutcicoensisnfgu lwliyt.h
in normal parameters.

System startup complete.
Welcome to PyOS v2.0!
```

Task 2.1 Write a Python script to simulate a basic system startup sequence.

```python
import time
import random
import sys

def print_slow(text, delay=0.05):
    """Print text slowly, like a terminal boot sequence."""
    for char in text:
        sys.stdout.write(char)
        sys.stdout.flush()
        time.sleep(delay)
    print()

def boot_sequence():
    print_slow("Initializing system startup...\n", 0.03)
    time.sleep(1)

    stages = [
        "Power On Self Test (POST)",
        "Checking hardware configuration",
        "Loading BIOS settings",
        "Initializing CPU cores",
        "Detecting memory modules",
        "Mounting system drives",
        "Loading kernel",
        "Starting system services",
        "Configuring network interfaces",
        "Launching user environment"
    ]

    for stage in stages:
        print_slow(f"[ OK ] {stage}", delay=0.02)
        time.sleep(random.uniform(0.4, 0.8))

    print_slow("\nSystem startup complete.", 0.04)
    print_slow("Welcome to PyOS v1.0!", 0.06)

if __name__ == "__main__":
    boot_sequence()
```

```
Initializing system startup...

[ OK ] Power On Self Test (POST)
[ OK ] Checking hardware configuration
[ OK ] Loading BIOS settings
[ OK ] Initializing CPU cores
[ OK ] Detecting memory modules
[ OK ] Mounting system drives
[ OK ] Loading kernel
[ OK ] Starting system services
[ OK ] Configuring network interfaces
[ OK ] Launching user environment

System startup complete.
Welcome to PyOS v1.0!
```

Task 2.2 : Use the multiprocessing module to create at least two child processes that perform dummy tasks.

```python
import multiprocessing
import time
import os

# Dummy task 1
def load_drivers():
    print(f"[Child 1 - PID {os.getpid()}] Loading hardware drivers...")
    time.sleep(2)
    print(f"[Child 1 - PID {os.getpid()}] Drivers loaded successfully.")

# Dummy task 2
def start_services():
    print(f"[Child 2 - PID {os.getpid()}] Starting system services...")
    time.sleep(3)
    print(f"[Child 2 - PID {os.getpid()}] Services started successfully.")

if __name__ == "__main__":
    print(f"[Parent - PID {os.getpid()}] Boot sequence initiated.\n")

    # Create child processes
    p1 = multiprocessing.Process(target=load_drivers)
    p2 = multiprocessing.Process(target=start_services)

    # Start child processes
    p1.start()
    p2.start()

    # Wait for child processes to finish
    p1.join()
    p2.join()

    print("\n[Parent] All startup tasks completed.")
    print("[Parent] System ready for login.")
```

```
[Parent - PID 3168] Boot sequence initiated.

[Child 1 - PID 3172] Loading hardware drivers...
[Child 2 - PID 3173] Starting system services...
[Child 1 - PID 3172] Drivers loaded successfully.
[Child 2 - PID 3173] Services started successfully.

[Parent] All startup tasks completed.
[Parent] System ready for login.
```

Task 2.3 Implement proper logging to track process start and end times

```python
import multiprocessing
import logging
import time
import os

# Configure logging
logging.basicConfig(
    level=logging.INFO,
    format="%(asctime)s - %(processName)s [PID %(process)d] - %(levelname)s - %(message)s",
    filename="process_log.log",
    filemode="w"
)

def task_1():
    logging.info("Task 1 started")
    time.sleep(2)   # Simulated work
    logging.info("Task 1 finished")

def task_2():
    logging.info("Task 2 started")
    time.sleep(3)   # Simulated work
    logging.info("Task 2 finished")


if __name__ == "__main__":
    logging.info("Parent process started")

    p1 = multiprocessing.Process(target=task_1, name="Worker-1")
    p2 = multiprocessing.Process(target=task_2, name="Worker-2")

    logging.info("Launching child processes...")
    p1.start()
    p2.start()

    p1.join()
    p2.join()

    logging.info("All child processes completed")
    logging.info("Parent process exiting")
```

```
2025-11-30 11:06:23,657 - MainProcess [PID 2604] - INFO - Parent process started
2025-11-30 11:06:23,657 - MainProcess [PID 2604] - INFO - Launching child processes...
2025-11-30 11:06:23,678 - Worker-1 [PID 2608] - INFO - Task 1 started
2025-11-30 11:06:23,679 - Worker-2 [PID 2609] - INFO - Task 2 started
2025-11-30 11:06:25,678 - Worker-1 [PID 2608] - INFO - Task 1 finished
2025-11-30 11:06:26,680 - Worker-2 [PID 2609] - INFO - Task 2 finished
2025-11-30 11:06:26,680 - MainProcess [PID 2604] - INFO - All child processes completed
2025-11-30 11:06:26,681 - MainProcess [PID 2604] - INFO - Parent process exiting
```

Task 2.4: Generate a log file (process log.txt) to reflect system-like behaviour

```python
import logging
import time
import random

# Configure logger to write to process_log.txt
logging.basicConfig(
    filename="process_log.txt",
    level=logging.INFO,
    format="%(asctime)s - %(levelname)s - %(message)s",
    filemode="w"
)

system_events = [
    "Initializing hardware modules",
    "Loading device drivers",
    "Detecting external peripherals",
    "Starting system daemons",
    "Activating security modules",
    "Establishing network connection",
    "Checking update repository",
    "Launching desktop environment",
    "Background services running",
    "System fully operational"
]

logging.info("=== System Startup Initiated ===")

# Log each event with random time delay for realism
for event in system_events:
    time.sleep(random.uniform(0.4, 1.1))
    logging.info(event)

logging.info("=== System Startup Completed ===")
```

```
2025-11-30 11:08:32,344 - INFO - === System Startup Initiated ===
2025-11-30 11:08:33,417 - INFO - Initializing hardware modules
2025-11-30 11:08:33,946 - INFO - Loading device drivers
2025-11-30 11:08:34,423 - INFO - Detecting external peripherals
2025-11-30 11:08:35,370 - INFO - Starting system daemons
2025-11-30 11:08:36,331 - INFO - Activating security modules
2025-11-30 11:08:36,994 - INFO - Establishing network connection
2025-11-30 11:08:37,536 - INFO - Checking update repository
2025-11-30 11:08:38,355 - INFO - Launching desktop environment
2025-11-30 11:08:39,288 - INFO - Background services running
2025-11-30 11:08:39,730 - INFO - System fully operational
2025-11-30 11:08:39,730 - INFO - === System Startup Completed ===
```

Task 3.1 CPU Scheduling with Gantt Chart

```python
processes = []
n = int(input("Enter number of processes: "))
for i in range(n):
    bt = int(input(f"Enter Burst Time for P{i+1}: "))
    pr = int(input(f"Enter Priority (lower number = higher priority) for P{i+1}: "))
    processes.append((i+1, bt, pr))
processes.sort(key=lambda x: x[2])
wt = 0
total_wt = 0
total_tt = 0
print("\nPriority Scheduling:")
print("PID\tBT\tPriority\tWT\tTAT")
for pid, bt, pr in processes:
    tat = wt + bt
    print(f"{pid}\t{bt}\t{pr}\t\t{wt}\t{tat}")
    total_wt += wt
    total_tt += tat
    wt += bt
print(f"Average Waiting Time: {total_wt / n}")
print(f"Average Turnaround Time: {total_tt / n}")

```

```
Enter number of processes: 2
Enter Burst Time for P1: 2
Enter Priority (lower number = higher priority) for P1: 2
Enter Burst Time for P2: 4
Enter Priority (lower number = higher priority) for P2: 1

Priority Scheduling:
PID     BT      Priority        WT      TAT
2       4       1               0       4
1       2       2               4       6
Average Waiting Time: 2.0
Average Turnaround Time: 5.0
```

Task 3.2 Sequential File Allocation.

```python
total_blocks = int(input("Enter total number of blocks: "))
block_status = [0] * total_blocks

n = int(input("Enter number of files: "))
for i in range(n):
    start = int(input(f"Enter starting block for file {i+1}: "))
    length = int(input(f"Enter length of file {i+1}: "))
    allocated = True
    for j in range(start, start+length):
        if j >= total_blocks or block_status[j] == 1:
            allocated = False
            break
    if allocated:
        for j in range(start, start+length):
            block_status[j] = 1
        print(f"File {i+1} allocated from block {start} to {start+length-1}")
    else:
        print(f"File {i+1} cannot be allocated.")

```

```
Enter total number of blocks: 3
Enter number of files: 5
Enter starting block for file 1: 3
Enter length of file 1: 2
File 1 cannot be allocated.
Enter starting block for file 2: 1
Enter length of file 2: 1
File 2 allocated from block 1 to 1
Enter starting block for file 3: 2
Enter length of file 3: 1
File 3 allocated from block 2 to 2
Enter starting block for file 4: 5
Enter length of file 4: 3
File 4 cannot be allocated.
Enter starting block for file 5: 4
Enter length of file 5: 2
File 5 cannot be allocated.
```

Task 3.3 Indexed File Allocation

```python
total_blocks = int(input("Enter total number of blocks: "))
block_status = [0] * total_blocks
n = int(input("Enter number of files: "))
for i in range(n):
    index = int(input(f"Enter index block for file {i+1}: "))
    if block_status[index] == 1:
        print("Index block already allocated.")
        continue
    count = int(input("Enter number of data blocks: "))
    data_blocks = list(map(int, input("Enter block numbers: ").split()))
    if any(block_status[blk] == 1 for blk in data_blocks) or len(data_blocks) != count:
        print("Block(s) already allocated or invalid input.")
        continue
    block_status[index] = 1
    for blk in data_blocks:
        block_status[blk] = 1
    print(f"File {i+1} allocated with index block {index} -> {data_blocks}")

```

```
Enter total number of blocks: 2
Enter number of files: 2
Enter index block for file 1: 0
Enter number of data blocks: 1
Enter block numbers: 1
File 1 allocated with index block 0 -> [1]
Enter index block for file 2: 1
Index block already allocated.
```

Task 3.4 Contiguous Memory Allocation

```python
def allocate_memory(strategy):
    partitions = list(map(int, input("Enter partition sizes: ").split()))
    processes = list(map(int, input("Enter process sizes: ").split()))
    allocation = [-1] * len(processes)

    for i, psize in enumerate(processes):
        idx = -1
        if strategy == "first":
            for j, part in enumerate(partitions):
                if part >= psize:
                    idx = j
                    break
        elif strategy == "best":
            best_fit = float("inf")
            for j, part in enumerate(partitions):
                if part >= psize and part < best_fit:
                    best_fit = part
                    idx = j
        elif strategy == "worst":
            worst_fit = -1
            for j, part in enumerate(partitions):
                if part >= psize and part > worst_fit:
                    worst_fit = part
                    idx = j
        if idx != -1:
            allocation[i] = idx
            partitions[idx] -= psize

    for i, a in enumerate(allocation):
        if a != -1:
            print(f"Process {i+1} allocated in Partition {a+1}")
        else:
            print(f"Process {i+1} cannot be allocated")

allocate_memory("first")
allocate_memory("best")
allocate_memory("worst")
```

```
Enter partition sizes: 10
Enter process sizes: 2
Process 1 allocated in Partition 1
Enter partition sizes: 5
Enter process sizes: 3
Process 1 allocated in Partition 1
Enter partition sizes: 1
Enter process sizes: 6
Process 1 cannot be allocated
```

Task 4.1: Batch Processing Simulation (Python)

```python
import subprocess
scripts = ['script1.py', 'script2.py', 'script3.py']
for script in scripts:
    print(f"Executing {script}...")
    subprocess.call(['python3', script])
```

Task 4.2: System Startup and Logging

```python
import multiprocessing
import logging
import time
logging.basicConfig(filename='system_log.txt', level=logging.INFO,
                    format='%(asctime)s - %(processName)s - %(message)s')
def process_task(name):
    logging.info(f"{name} started")
    time.sleep(2)
    logging.info(f"{name} terminated")
if __name__ == '__main__':
    print("System Booting...")
    p1 = multiprocessing.Process(target=process_task, args=("Process-1",))
    p2 = multiprocessing.Process(target=process_task, args=("Process-2",))
    p1.start()
    p2.start()
    p1.join()
    p2.join()
    print("System Shutdown.")

```

```
System Booting...
System Shutdown.
```

```
2025-11-30 11:21:11,703 - Process-1 - Process-1 started
2025-11-30 11:21:11,705 - Process-2 - Process-2 started
2025-11-30 11:21:13,704 - Process-1 - Process-1 terminated
2025-11-30 11:21:13,705 - Process-2 - Process-2 terminated

```

Task 4.3: System Calls and IPC (Python - fork, exec, pipe)

```python
import os
r, w = os.pipe()
pid = os.fork()
if pid > 0:
    os.close(r)
    os.write(w, b"Hello from parent")
    os.close(w)
    os.wait()
else:
    os.close(w)
    message = os.read(r, 1024)
    print("Child received:", message.decode())
    os.close(r)
```

```
Child received: Hello from parent
```

Task 4.4: VM Detection and Shell Interaction

```bash
#!/bin/bash

echo "===== SYSTEM INFORMATION REPORT ====="
echo "Hostname: $(hostname)"
echo "Operating System: $(uname -o 2>/dev/null)"
echo "Kernel Version: $(uname -r)"
echo "Architecture: $(uname -m)"
echo "Current User: $(whoami)"
echo "Uptime: $(uptime -p)"
echo "CPU Model: $(grep -m1 'model name' /proc/cpuinfo | cut -d ':' -f2)"
echo "Total Memory: $(grep MemTotal /proc/meminfo | awk '{print $2/1024 \" MB\"}')"
echo "Available Memory: $(grep MemAvailable /proc/meminfo | awk '{print $2/1024 \" MB\"}')"
echo "Disk Usage:"
df -h --total | grep total
echo "====================================="

```

input

```
main.bash: line 2: $'\r': command not found
===== SYSTEM INFORMATION REPORT =====
Hostname: Check
Operating System: GNU/Linux
Kernel Version: 6.8.0-1043-gcp
Architecture: x86_64
Current User: runner21
Uptime: up 1 hour, 7 minutes
CPU Model:  Intel(R) Xeon(R) CPU @ 2.80GHz
awk: 1: unexpected character '\'
awk: line 1: runaway string constant " MB\"} ...
Total Memory:
awk: 1: unexpected character '\'
awk: line 1: runaway string constant " MB\"} ...
Available Memory:
Disk Usage:
=====================================
```

```python
import subprocess

def detect_vm():
    indicators = []

    # Check DMI info for known VM vendors
    try:
        output = subprocess.check_output("dmidecode -s system-product-name", shell=True, text=True, stderr=subprocess.DEVNULL).lower()
        indicators.append(output)
    except:
        pass

    try:
        output2 = subprocess.check_output("dmidecode -s system-manufacturer", shell=True, text=True, stderr=subprocess.DEVNULL).lower()
        indicators.append(output2)
    except:
        pass

    vm_keywords = ["virtualbox", "vmware", "qemu", "kvm", "hyper-v", "xen", "parallels"]

    vm_detected = any(keyword in " ".join(indicators) for keyword in vm_keywords)

    if vm_detected:
        print("⚠ Virtual Machine environment detected")
        print("Matched info:", indicators)
    else:
        print("✓ Physical machine detected (no VM signatures found)")

if __name__ == "__main__":
    detect_vm()

```

input

```
✓ Physical machine detected (no VM signatures found)
```