Assignment 2

Part A

1. Hello, World!

2. Variable name will have value Productive

3. Create a new file named file.txt

4. List all files including those that are hidden (starting from.)

5. Remove/delete the file named file.txt

6. Copy file named file1.txt and name the new file as file2.txt

7. Move the file named file.txt to a directory named directory.

8. It will change the file permissions of file named script.sh: Owner, Group and Others to (Read, Write, Execute) , (Read, not write, Execute) and (Read, not write, Execute) respectively.

9. Search for the term “pattern” in the file named file.txt

10. It will stop the process PID

11. It will create a directory mydir, then open it, and create a file named file.txt. It will then edit the file.txt file to print a statement “Hello, World!” and finally print the output of file named file.txt.

12. It will search for files with “.txt” in their name and list them.

13. It will display only the unique lines of files named file1.txt and file2.txt

14. It will search and list all the files starting with “d”.

15. It will recursively search for the term “pattern” in the directory named directory.

16. It will print only those lines that are duplicate in the files named file1.txt and file2.txt.

17. It will change the file permissions of the file named file.txt: Owner, Group and Others to (Read, Write, not execute) , (Read, not write, not execute) and (Read, not write, not execute) respectively.

18. It will recursively copy the directory named “source_directory” to the directory named “destination_directory”.

19. It will find a file with name “.txt” in the search directory.

20. It will provide the Execute permission to Owner of the file named file.txt.

21. It will print the value of the variable named PATH.

Assignment 2

Part B

1. True
2. True
3. False
4. True
5. True
6. True
7. True
8. False

The following are the Incorrect Commands: 1, 2, 4.

Assignment 2

Part C

1. echo “Hello, World!”

2. name = “CDAC Mumbai”.
   echo $name

3. num=0
   echo Write a number:
   read num
   echo The number is: $num

4. a=5
   b=3
   sum= $((a+b))
   echo The Sum of the numbers is: $sum

5. num=0
   echo Please give a number
   read num
   if ((num % 2 == 0))
   then
   echo The Number is even
   else
   echo The Number is odd
   fi

6. a=0
   for ($a -lt 6)
   echo $a
   done

7. b=0
   while ($b -lt 6)
   echo $b
   $b=$b+1
   done

8. if [-f “file.txt”];

```
then
echo “File exists”
else
echo “ File does not exist”
fi
```

9.
```
num=0
echo “Enter a number”
read num
if [$num -gt 10]
then
echo “The number is greater than 10.”
else
echo “The number is not greater than 10.”
Fi
```

10.
```
a=0
 b=0
 echo “Multiplication Table from numbers 1 to 5”
 for a in {1..5};
 do
        for b in {1..5};
        do
        echo “$a x $b = $((a*b))”
        done
 done
```

11.
```
num=0
sq=0
while true;
do
echo “Enter a number:”
read num
if [$num – lt 0];
then
echo “It is a Negative number.”
Break
fi
sq=$((num * num))
echo “The square of the number $num is: $sq”
done
```

Assignment 2

Part D

1. What is an operating system, and what are its primary functions?

An operating system (OS) is system software that manages computer hardware and software resources.
Primary functions:

- Process management
- Memory management
- File system management
- Device management
- User interface management

2. Explain the difference between a process and a thread.

- A **process** is an unit of a big program, and has it's own memory space.
- A **thread** is an unit within a process, sharing memory and resources with other threads of the same process.

3. What is virtual memory, and how does it work?

Virtual memory extends RAM by using secondary storage, allowing processes to execute larger applications than available physical memory. It works using **paging**, where memory is divided into pages that can be swapped between RAM and disk.

4. Describe the difference between multiprogramming, multitasking, and multiprocessing.

- **Multiprogramming:** Multiple programs are loaded in memory to increase CPU utilization.
- **Multitasking:** Multiple tasks (processes) run concurrently by sharing CPU resources.
- **Multiprocessing:** Multiple CPUs or cores execute multiple processes simultaneously.

5. What is a file system, and what are its components?

A file system organizes and manages data on storage devices.
Components:

- Files
- Directories
- File attributes
- Metadata

6. What is a deadlock, and how can it be prevented?

A deadlock occurs when processes hold resources and wait for others to release their resources, causing a circular wait.
Prevention methods:

- Deadlock ignorance
- Deadlock avoidance (e.g., Banker's algorithm)
- Deadlock detection and recovery

7. Explain the difference between a kernel and a shell.

- **Kernel:** Core OS component managing hardware and system resources.
- **Shell:** User interface for accessing OS services, e.g., Bash, PowerShell.

8. What is CPU scheduling, and why is it important?

CPU scheduling decides which process gets CPU time, optimizing efficiency..

9. How does a system call work?

A system call is a request from a user program to the kernel for operations like file acces.

10. What is the purpose of device drivers in an operating system?

Device drivers act as a medium between hardware and the OS, enabling communication with Input- Output devices.

11. Explain the role of the page table in virtual memory management.

The page table maps virtual addresses to physical addresses, enabling efficient memory access.

12. What is thrashing, and how can it be avoided?

Thrashing occurs when excessive paging reduces performance.

Avoidance:

- Increasing RAM
- Using algorithms
- Page replacement policies

13. Describe the concept of a semaphore and its use in synchronization.

A semaphore is a synchronization method used to manage concurrent process access to shared resources, preventing race conditions.

14. How does an operating system handle process synchronization?

Using synchronization mechanisms like semaphores and mutexes to prevent race conditions.

15. What is the purpose of an interrupt in operating systems?

Interrupts allow the CPU to handle urgent events by temporarily pausing the current process and executing an interrupt.

16. Explain the concept of a file descriptor.

A file descriptor is an integer representing an open file or I/O resource in a process.

17. How does a system recover from a system crash?

Recovery techniques include a restart, log-based recovery, and restoring from backups.

18. Describe the difference between a monolithic kernel and a microkernel.

- **Monolithic kernel:** All OS services run in kernel space (e.g., Linux).
- **Microkernel:** Only essential services run in kernel space; others run in user space (e.g., Minix).

19. What is the difference between internal and external fragmentation?

- **Internal fragmentation:** Wasted memory inside allocated memory blocks.
- **External fragmentation:** Free memory block exists but is non-contiguous.

20. How does an operating system manage I/O operations?

Using I/O schedulers, buffering, caching, and device drivers.

21. Explain the difference between preemptive and non-preemptive scheduling.

- **Preemptive:** OS can interrupt a process (e.g., Round-Robin).
- **Non-preemptive:** Process runs until completion (e.g., First Come First Serve).

22. What is round-robin scheduling, and how does it work?

A preemptive scheduling algorithm where each process gets a fixed time slice (quantum) before moving to the next.

23. Describe the priority scheduling algorithm.

Processes are scheduled based on priority; higher priority executes first.

24. What is the shortest job next (SJN) scheduling algorithm, and when is it used?

Processes with the shortest execution time are scheduled first, reducing average waiting time.

25. Explain the concept of multilevel queue scheduling.

Processes are divided into different queues based on priority or type, each having its own scheduling algorithm.

26. What is a process control block (PCB), and what information does it contain?

A PCB stores process-related information, including process ID, state, registers, and memory details.

27. Describe the process state diagram and transitions between states.

States: New → Ready → Running → Waiting → Terminated.

28. How does a process communicate with another process in an operating system?

Using IPC mechanisms like pipes, signal queues and shared memory.

29. What is process synchronization, and why is it important?

Ensures correct execution by preventing race conditions.

30. Explain the concept of a zombie process and how it is created.

A zombie process has completed execution but still has an entry in the process table because its parent hasn't read its exit status.

31. Describe the difference between internal and external fragmentation.

Internal Fragmentation:

- Occurs inside allocated memory blocks when the allocated space is larger than required.
- Example: A process needs 30 KB, but the OS allocates a 32 KB block, wasting 2 KB.
- Use paging or allocate memory dynamically.

External Fragmentation:

- Happens when free memory is scattered across the system, making it unusable for large allocations.
- Example: Multiple small free memory gaps exist, but no single large block is available.
- Compaction or use paging/segmentation.

32. What is demand paging, and how does it improve memory management efficiency?

Only loads required pages into memory, reducing initial loading time.

33. Explain the role of the page table in virtual memory management.

A page table maps virtual addresses to physical addresses in paging-based memory systems.

Functions:
Keeps track of page numbers and their corresponding frame numbers in RAM.
Enables address translation (virtual → physical).
Stores protection bits (read/write permissions).
Helps in demand paging (loading pages only when needed).

34. How does a memory management unit (MMU) work?

Translates virtual addresses to physical addresses using a page table.

35. What is thrashing, and how can it be avoided in virtual memory systems?

Thrashing occurs when excessive page swapping (paging in and out) reduces CPU efficiency. The system spends more time managing memory than executing processes.

Causes:

- Too many processes in memory.
- Insufficient RAM.
- Poor page replacement strategy.

Solutions:

- Working Set Model: Keep only frequently used pages in memory.
- Increase RAM or use better page replacement algorithms (LRU, LFU).

36. What is a system call, and how does it facilitate communication between user programs and the OS?

A system call allows user applications to request services from the operating system kernel.

Example Actions via System Calls:

- File operations: open(), read(), write()
- Process management: fork(), exec(), exit()

- Memory management: mmap()
- Device I/O: ioctl(), write()

37. Describe the difference between a monolithic kernel and a microkernel.

Monolithic Kernel: Handles everything (memory, process, drivers) inside the kernel itself.

Microkernel: Only manages core functions (processes, scheduling), while other services run in user space.

38. How does an operating system handle I/O operations?

I/O operations involve communication between the CPU, memory, and devices (disk, keyboard, network, etc.).

OS Role in I/O:

1. Uses Device Drivers to interact with hardware.
2. Uses Interrupts to handle events efficiently.
3. Uses Buffering & Caching for performance optimization.

39. Explain the concept of a race condition and how it can be prevented.

A race condition occurs when multiple processes modify shared data simultaneously, leading to unpredictable results. **Prevented by:** Mutexes, semaphores.

40. Describe the role of device drivers in an operating system.

A device driver is a software component that allows the OS to interact with hardware devices like printers, USBs, and network adapters.

Roles:

- Acts as a medium between the OS and hardware.
- Handles I/O operations (e.g., reading a USB drive).
- Uses interrupts for efficiency.
- Ensures hardware compatibility with the OS.

Example:
A printer driver converts a document into instructions that are only related to printers so that the OS can communicate correctly.

41. What is a zombie process, and how does it occur?

A zombie process is a process that has terminated but still exists in the process table because its parent hasn't read its exit status.

How It Happens:

1. A child process completes execution and enters a "defunct" state.
2. The parent fails to call wait() or waitpid(), leaving the child's PID occupied.
3. The zombie remains in the process table until the system reboots or the parent collects the exit status.

How to Prevent Zombie Processes:

- The parent should use wait() or waitpid() to remove the child's entry.

42. Explain the concept of an orphan process.

A child process that continues running after its parent terminates.

Handled by: The init process in Unix-like OS.

43. What is the relationship between a parent process and a child process in the context of process management?

A parent process is a process that creates another process using the fork() system call. The newly created process is called a child process.

The child process receives certain qualities from the parent, such as user ID. The parent can wait for the child to complete or continue execution independently.

44. How does the fork() system call work in creating a new process in Unix-like operating systems?

The fork() system call creates a new process by duplicating the calling process.

- The parent process calls fork().
- The OS creates a child process with the same memory layout as the parent.

- fork() returns 0 to the child and the child’s PID to the parent.

45. Describe how a parent process can wait for a child process to finish execution.

A parent process can use the wait() or waitpid() system calls to pause execution until the child process completes.

This prevents zombie processes by allowing the parent to collect the child's exit status.

46. What is the significance of the exit status of a child process in the wait() system call?

The exit status allows the parent to know how the child process terminated.

- A return value of 0 means the child terminated successfully.
- A non-zero value indicates an error or specific exit condition.

47. How can a parent process terminate a child process in Unix-like operating systems?

A parent can use the kill() system call with the child's process ID (PID):

Alternatively, a parent can check and terminate orphaned children using SIGCHLD handling.

48. Explain the difference between a process group and a session in Unix-like operating systems.

- A process group consists of one or more processes that can receive the same signals collectively. They are identified by a group ID.
- A session is a collection of one or more process groups.

A session leader can create new process groups for better control.

49. Describe how the exec() family of functions is used to replace the current process image with a new one.

The exec() family (execve(), execlp(), execvp()) replaces the current process with a new program.

- The PID remains the same, but the memory image is replaced.
- Unlike fork(), exec() does not return unless an error occurs.

50. What is the purpose of the waitpid() system call in process management? How does it differ from wait()?

waitpid() allows a parent to wait for a specific child process, unlike wait(), which waits for any child.

- wait() waits for any child, while waitpid() waits for a specific PID.
- waitpid() can use flags like WNOHANG to return immediately if the child hasn't exited.

51. How does process termination occur in Unix-like operating systems?

A process terminates when it:

- Calls exit.
- Receives a termination signal.
- Encounters an unrecoverable error (e.g., segmentation fault).
- Completes execution normally.

The OS cleans up resources and, if the process is a zombie, waits for the parent to acknowledge termination.

52. What is the role of the long-term scheduler in the process scheduling hierarchy? How does it influence the degree of multiprogramming in an operating system?

The long-term scheduler controls the number of processes admitted to the system.

- It selects which process move from disk to ready queue.
- If too many processes are admitted, thrashing may occur.

53. How does the short-term scheduler differ from the long-term and medium-term schedulers in terms of frequency of execution and the scope of its decisions?

Scheduler Type	Execution Frequency	Role
Long-term	Infrequent	Controls admission of new processes

Scheduler Type	Execution Frequency	Role
Medium-term	Occasionally	Swaps processes in/out of memory
Short-term	Very frequent	Decides which process gets CPU next

The **short-term scheduler** runs every few milliseconds, switching between processes rapidly.

54. Describe a scenario where the medium-term scheduler would be invoked and explain how it helps manage system resources more efficiently.

A scenario where the medium-term scheduler is invoked:

- If RAM is full, and a background process is inactive, it may be swapped to disk to free memory.
- When resources are available, the process is swapped back into RAM.

This helps by:
Reducing memory pressure
Preventing thrashing
Optimizing CPU-bound vs. I/O-bound process execution

Assignment 2

Part E

Date: __/__/____

Gantt Chart: P_1 (0–5), P_2 (5–8), P_3 (8–14)

(1) First-Come First-Served (FCFS) scheduling:

PID	Arrival Time	Burst Time	Response Time	Waiting Time	Turn Around Time
P_1	0	5	0	0	5
P_2	1	3	5	4	7
P_3	2	6	8	6	12

Ans. Average Waiting Time $= \frac{0+4+6}{3} = \frac{10}{3} = 3.33$

(2) Shortest Job First (SJF):

PID	AT	BT	RT	WT	TAT
P_1	0	3	0	0	3
P_2	1	5	4	8	13
P_3	2	1	3	1	2
P_4	3	4	4	1	5

Gantt Chart: P_1 (0–3), P_3 (3–4), P_4 (4–9), P_2 (9–14)

Ans. Average Turn Around Time : $\frac{23}{4}$

$\Rightarrow 5.75$

Date: __/__/____

(3) Priority Scheduling:

PID	AT	BT	Priority	RT	WT	TAT
P_1	0	6	3	0	0	6
P_2	1	4	1	6	5	9
P_3	2	7	4	12	10	17
P_4	3	2	2	10	7	9

Gantt Chart

P_1	P_2	P_4	P_3
0	6	10	12	19

Ans. Average Waiting time = 5.50

(4) Round Robin Scheduling:

PID	AT	BT	RT	WT	TAT
P_1	0	4	0	0	10
P_2	1	5	2	1	15
P_3	2	2	4	2	4
P_4	3	3	6	3	10

Gantt Chart

P_1	P_2	P_3	P_4	P_1	P_2	P_4	P_2
0	2	4	6	8	10	12	13	15

Ans. Average Turn Around Time $= \frac{39}{4} = 9.75$

or $\frac{40}{4} = 10.$

(5)

```
x=5
{
  x=$((x+1))
  echo "Child : x = $x"
} &
x= $((x+1))
echo "Parent : x = $x"
wait
```

Parent : x=6
Child : x=6