

* 1. **.:** 2023-24 **Class: S.Y.B.Tech Sub:** System Fundamentals

**System Fundamentals Experiment List**

Explore the internal commands of Linux and Write shell scripts to do the following:

* + 1. **Display top 10 processes in descending order**

**CODE:-** ps aux --sort=-%mem | head -n 11

* + 1. **Display processes with highest memory usage.**

**CODE:-** ps aux --sort=-%mem | head -n 2

* + 1. **Display current logged in user and logname.**

**CODE:- echo "Logged in user: $(whoami), Logname: $(logname)"**

* + 1. **Display current shell, home directory, operating system type, current path setting, current working directory.**

**CODE:-** echo “Shell: $SHELL”

echo “Home directory: $HOME”

echo “Operating system type: $(uname -o)”

echo “Current path: $PATH”

echo “Current working directory: $PWD”

* + 1. **Display OS version, release number, kernel version..**

**CODE:-** lsb\_release -a

uname -r

# Write a command to display the first 15 columns from each line in the file

# CODE:-

# head -15 xyz.txt

* + 1. **cut specified columns from a file and display them**

**CODE:-**

cut -c 1,3,4 xyz.txt

* + 1. **Sort given file ignoring upper and lower case**

**CODE:-**

sort -f xyz.txt

* + 1. **Displays only directories in current working directory.**
    2. **copying files from one place to another,**
    3. **moving files from one place to another.**
    4. **Removing specific directory with various options**
    5. **list the numbers of users currently login in the system and then sort it.**
    6. **Merge two files into one file**
    7. **changes the access mode of one file**
    8. **display the last ten lines of the file.**
    9. **to locate files in a directory and in a subdirectory.**
    10. **This displays the contents of all files having a name starting with ap followed by any number of characters.**
    11. **Rename any file aaa to aaa.aa1, where aa1 is the user login name.**

**CODES:-**

**9.**  ls -d \*/

**10.** cp

**11.** mv

**12.** rmdir dir\_name

**13.** who | awk '{print $1}' | sort | uniq -c | sort -nr

**14.** echo "Enter the first file name"

read file1

echo "Enter the second file name"

read file2

echo "Enter the output file name"

read output\_file

cat “$file1” “file2” > “$output\_file”

echo "files merged successfully"

**15.** echo "Enter the file name"

read file

echo "Enter the new access mode (e.g., 755):"

read new\_mode

chmod “$new\_mode” “$file”

echo "access mode changed successfully"

**16.** echo "Enter the file name"

read file

tail -n 10 “$file”

**17**.

echo "Enter the directory to search:"

read search\_directory

echo "Enter the file name to locate:"

read file\_name

find "$search\_directory" -type f -name "$file\_name**"**

**18. find /path/to/directory -type f -regex ".\*ap.\*"**

**19.**

find /path/to/directory -type f -name "aaa" -execdir rename 's/(aaa)$/$1.'"$(whoami)"'.aa1/' {} \;

echo "Enter the file to rename (aaa): "

read file

new\_name="${file).${USER}"

mv "$file" "$new\_name"

echo "File renamed to $new\_name."

Illustrate the use of sort, grep, awk, etc.

* + 1. **Write a command to search the word ‘picture’ in the file and if found, the lines containing it would be displayed on the screen.**

**CODE:-** grep 'picture' filename.txt

* + 1. **Write a command to search for all occurrences of ‘Rebecca’ as well as ‘rebecca’ in file and display the lines which contain one of these words.**

**CODE:-** grep -i 'Rebecca\|rebecca' filename.txt

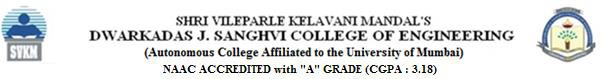
* + 1. **Write a command to search all four-letter words whose first letter is a ‘b’ and last letter, a ‘k’.**

**CODE:-** grep -w '^b..k$' filename.txt

* + 1. **Write a command to see only those lines which do not contain the search patterns**

**CODE:-** grep -v 'pattern1\|pattern2' filename.txt

* + 1. **Implement Booth’s multiplication algorithm.**
    2. **Implement Restoring division algorithm.**
    3. **Implement Non-Restoring division algorithm.**
    4. **Implement fully associative memory mapped cache organization.**
    5. **Implement various LRU cache/page replacement policy**



**A.Y.:** 2023-24 **Class: S.Y.B.Tech Sub:** System Fundamentals

* + 1. **Implement various optimal cache/page replacement policy**
    2. **Implement various FIFO cache/page replacement policy**
    3. **Implement FCFS CPU scheduling algorithm.**
    4. **Implement SJF CPU scheduling algorithm.**
    5. **Implement Non Prremptive Priority CPU scheduling algorithm.**
    6. **Implement Prremptive Priority CPU scheduling algorithm.**
    7. **Implement SRTF CPU scheduling algorithm.**
    8. **Implement Round Robin CPU scheduling algorithm.**
    9. **Implement Best Fit Memory allocation policy.**

**﻿**

def best fit (block\_size, process\_size):

m = len(block\_size)

n = len(process\_size)

allocation = [-1] \* n

for i in range(n):

best\_fit\_idx = -1

for j in range(m):

if block\_size[j] >= process\_size[i]:

if best\_fit\_idx == -1 or block\_size[j] < block\_size[best\_fit\_idx]:

best\_fit\_idx = j

if best\_fit\_idx != -1:

allocation[i] = best\_fit\_idx

block\_size[best\_fit\_idx]-= process\_size[i]

print("Process No. \tProcess Size\tBlock No.")

for i in range(n):

print(f"{i+1}\t\t{process\_size[i]}\t\t{allocation[i]+1 if allocation[i] != -1 else 'Not Allocated'}")

# Example usage:

block\_size = [100, 500, 200, 300, 600]

process\_size = [212, 417, 112, 426]

best\_fit (block\_size, process\_size)

* + 1. **Implement First Fit Memory allocation policy.**

def first\_fit (memory\_blocks, process\_sizes):

allocation [-1] \* len(process\_sizes)

for i in range(len(process\_sizes)):

for j in range(len (memory\_blocks)):

if memory\_blocks[j] >= process\_sizes[i]:

allocation[i] = j

memory\_blocks[j] -= process\_sizes[i]

break

print("Process No. \tProcess Size\tBlock No.")

for i in range(len(process\_sizes)):

print (f"{i + 1} \t\t{process\_sizes[i]}\t\t", end="")

if allocation[i] != -1:

print (allocation[i] + 1)

else:

print("Not Allocated")

# Example usage:

memory\_blocks= [100, 500, 200, 300, 600]

process\_sizes= [212, 417, 112, 426]

first\_fit (memory\_blocks, process\_sizes)

* + 1. **Implement Worst Fit Memory allocation policy.**

def worst\_fit (memory\_blocks, process\_sizes):

allocation [-1] \* len(process\_sizes)

for i in range(len(process\_sizes)):

worst\_index = -1

for j in range(len(memory\_blocks)):

if memory\_blocks[j] >= process\_sizes[i]:

if worst\_index == -1 or memory\_blocks[j] > memory\_blocks [worst\_index]:

worst\_index = j

if worst\_index != -1:

allocation[i] = worst\_index

memory\_blocks[worst\_index] -= process\_sizes[i]

print("Process No.\tProcess Size\tBlock No.")

for i in range(len(process\_sizes)):

print(f"{i + 1}\t\t{process\_sizes[i]}\t\t", end="")

if allocation[i] != -1:

print(allocation[i] + 1)

else:

print("Not Allocated")

# Example usage:

memory\_blocks = [100, 500, 200, 300, 600]

process\_sizes = [212, 417, 112, 426]

worst fit (memory\_blocks, process\_sizes)

* + 1. **Implement Producer -Consumer problem with Semaphore.**
    2. **Implement order scheduling in supply chain using Banker’s Algorithm**

﻿class BankerAlgorithm:

def \_\_init\_\_(self, processes, resources):

self.processes=processes

self.resources = resources

self.max\_claim = [[5, 5, 7], [3, 2, 2], [9, 0, 2], [2, 2, 2], [4, 3, 3]]

self.allocation =[[0, 1, 0], [2, 0, 0], [3, 0, 2], [2, 1, 1], [0, 0, 2]]

self.need=[[0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0]]

self.safe\_sequence = []

self.work = resources.copy()

self.finish = [False] \* processes

def calculate\_need\_matrix(self):

for i in range(self.processes):

for j in range(self.resources):

self.need[i][j]= self.max\_claim[i][j] - self.allocation[i][j]

def is\_safe\_state(self):

for i in range(self.processes):

if not self.finish[i] and all (need <= self.work for need in self.need[i]):

self.work = [work + allocation for work, allocation in zip(self.work, self.allocation[i])] self.safe\_sequence.append(i)

self.finish[i] = True

return self.is\_safe\_state()

return all(self.finish)

def run(self):

self.calculate\_need\_matrix()

if self.is\_safe\_state():

print("Safe state found.")

print("Safe Sequence:", self.safe\_sequence)

else:

print("Unsafe state. No safe sequence found.")

# Example usage:

banker

=

BankerAlgorithm (processes=5, resources=3)

banker.run()

* + 1. **Implement FIFO Disk Scheduling Algorithms.**

def fifo\_disk\_scheduling (requests, head):

total\_head\_movements = 0

current head = head

Copy code

for request in requests:

total\_head\_movements += abs(current\_head-request)

current\_head = request

return total\_head\_movements

# Example usage:

requests = [98, 183, 37, 122, 14, 124, 65, 67]

initial\_head = 53

result = fifo\_disk\_scheduling (requests, initial\_head)

print("FIFO Disk Scheduling Algorithm: Total head movements = {result}

* + 1. **Implement SSTF Disk Scheduling Algorithms.**

def sstf\_disk\_scheduling (requests):

seek\_count = 0

current\_track = 0

while requests:

closest\_request = min(requests, key-lambda x: abs(x - current\_track))

seek\_count += abs(current\_track - closest\_request)

current\_track = closest\_request requests.remove(closest\_request)

return seek\_count

# Example Usage

requests = [98, 183, 37, 122, 14, 124, 65, 67]

sstf\_result = sstf\_disk\_scheduling (requests)

print (f"SSTF Disk Scheduling Seek Count: {sstf\_result}")

* + 1. **Implement SCAN Disk Scheduling Algorithms.**

def scan\_disk\_scheduling(requests, start\_direction="left"):

seek\_count = 0

current\_track = 0

if start\_direction=="left":

requests.sort()

else:

requests.sort(reverse=True)

for request in requests:

seek\_count += abs(current\_track - request) current\_track

return seek\_count

# Example Usage

=

request

requests = [98, 183, 37, 122, 14, 124, 65, 67]

scan\_result = scan\_disk\_scheduling (requests, start\_direction="left")

print (f"SCAN Disk Scheduling Seek Count: {scan\_result}")

* + 1. **Implement C-SCAN Disk Scheduling Algorithms.**

**﻿**def c\_scan\_disk\_scheduling (requests, start\_direction="left"):

seek\_count = 0

current\_track = 0

if start\_direction== "left":

requests.sort()

else:

requests.sort (reverse=True)

for request in requests:

seek\_count += abs(current\_track - request)

current\_track = request

# Move to the other end of the disk

seek\_count += abs(current\_track - (0 if start\_direction == "left" else max(requests)))

return seek\_count

# Example Usage

requests = [98, 183, 37, 122, 14, 124, 65, 67]

c\_scan\_result = c\_scan\_disk\_scheduling (requests, start\_direction="left")

print (f"C-SCAN Disk Scheduling Seek Count: {c\_scan\_result}")

* + 1. **Implement Look Disk Scheduling Algorithms.**
    2. **Implement Look Disk Scheduling Algorithms.**

Implement Multithreading to create child processes using fork() system call.

* + 1. **Program where parent process sorts array elements in descending order and child process sorts array elements in ascending order.**
    2. **Program where parent process Counts number of vowels in the given sentence and child process will count number of words in the same sentence. The above programs should use UNIX calls like fork, exec and wait. And also show the orphan and zombie states**
    3. **Write Shell script to copy files from one folder to another**
    4. **Write Shell script Count number of words, characters and lines.**
    5. **Write Shell script To describe files in different format.**
    6. **Write Shell script to find factorial of given number using bash script**
    7. **Display first 10 natural numbers using bash script**
    8. **Display Fibonacci series using bash script**
    9. **Find given number is prime or nor using bash script**
    10. **Write shell script to finding biggest of three numbers**
    11. **Write shell script to reversing a number**
    12. **Write shell script find Sum of individual digits (1234 -&gt; 1+2+3+4=10)**

**CODES:-**

48. # Function to sort array in descending order

sort descending() {

arr=("$@")

n=${#arr[@]}

for ((i = 0; i < n-1; i++)); do

}

for ((j

=

0; j < n-i-1; j++)); do

if ((arr[j] < arr[j+1])); then

temp=${arr[j]}

arr[j]=${arr[j+1]}

arr[j+1]=$temp

fi

done

done

echo "${arr[@]}"

# Function to sort array in ascending order

sort ascending() {

arr=("$@")

n=${#arr[@]}

for ((i = 0; i < n-1; i++)); do

for ((j = 0; j < n-i-1; j+)); do

if ((arr[j] > arr[j+1])); then

fi

temp=${arr[j]}

arr[j]=${arr[j+1]} arr[j+1]=$temp

done

done

}

echo "${arr[@]}"

#Main script

echo "Enter array elements separated by spaces:"

read a array

# Forking a child process

if [ "Spid" -eq 0 ]; then

# Inside child process

echo "Child process (ascending): $(sort\_ascending "$(array[@]}")" else

# Inside parent process

wait Spid

echo "Parent process (descending): $(sort\_descending "${array[@]}")"

fi

49.

echo "Enter a sentence:"

read sentence

# Function to count vowels

count\_vowels() {

echo "$1" | tr -cd 'aeiouAEIOU’ | wc -c

}

# Forking a child process

if [ "$pid" -eq 0 ]; then

# Inside child process

exec echo "Child process (word count): $(echo $sentence | wc -w)"

else

# Inside parent process

wait $pid

echo "Parent process (vowel count): $(count\_vowels "$sentence")"

fi

50.

echo "Enter the Source destination"

read source\_folder

echo "Enter the Destination destination"

read destination\_folder

cp -r "$source\_folder"/\* "$destination\_folder"

echo "File copied successfully"

51. echo "Enter the File name"

read file\_name

word\_count=$(wc -w < "$file\_name")

char\_count=$(wc -c < "$file\_name")

line\_count=$(wc -l < "$file\_name")

echo "Word Count: $word\_count"

echo "Character Count: $char\_count"

echo "Line Count: $line\_count"

52.

echo "Enter the file name"

read file\_name

file\_info=$(file "$file\_name")

echo "File Information: $file\_info"

53.

echo "Enter the number "

read num

factorial=1

for((i=1;i<=num;i++))do

 factorial=$((factorial\*i))

done

echo "Factorial of $num is $factorial"

54.

 echo "Enter the last Natural Number "

read nn

for((i=1;i<=nn;i++))do

 echo "$i"

done

55. echo "Enter the number"

read num

a=0

b=1

echo "Fibonacci series is "

echo -n "$a $b "

for((i=3;i<=num;i++))do

 c=$((a+b))

 echo -n "$c "

 a=$b

 b=$c

done

56. echo "Enter the number"

read num

is\_prime=true

if [ "$num" -lt 2 ]; then

  is\_prime=false

fi

for((i=2;i<=num/2;i++)) do

 if [ $((num%i)) -eq 0 ]; then

  is\_prime=false

  break

 fi

done

 if [ "$is\_prime" = true ]; then

  echo "Entered number is a prime number"

 else

  echo "Entered number is not a prime number"

 fi

57. echo "Enter three numbers :"

read a

read b

read c

max=$a

if [ $b -gt $max ]; then

 max=$b

fi

if [ $c -gt $max ]; then

 max=$c

fi

echo "The largest number is $max"

58. echo "Enter the number:"

read num

reverse\_number=0

original=$num

while [ $num -gt 0 ]; do

 digit=$((num % 10))

 reverse\_number=$((reverse\_number \* 10 + digit))

 num=$((num / 10))

done

echo "The reverse of $original is $reverse\_number"

59. echo "Enter the number:"

read num

original=$num

sum=0

while [ $original -gt 0 ]; do

 digit=$((original % 10))

 sum=$((sum + digit))

 original=$((original / 10))

done

echo "The sum of individual digits is $sum"