

Assignment#1

Operating Systems

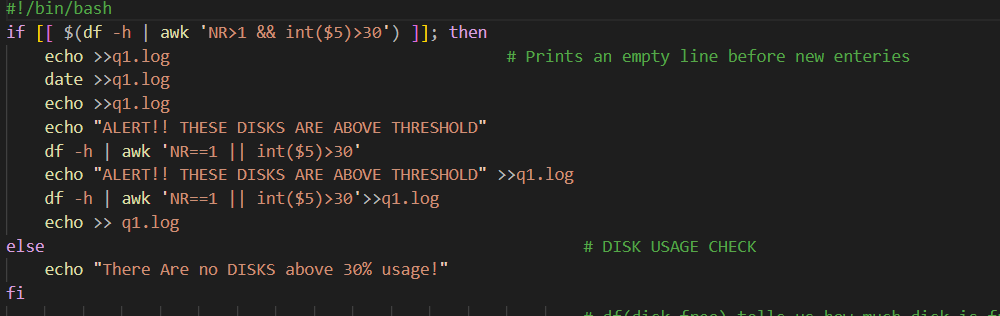


March 11, 2024

AHmad Abdullah(i22-1609)

## Question#1

### Disk Usage Check



# df(disk free) tells us how much disk is free and

# -h presents it in human-readable form

# '|' is a pipe which takes the output of the left command

# and makes it the input of the command on the right

# awk is used for text filtering and NR>1 ensures that

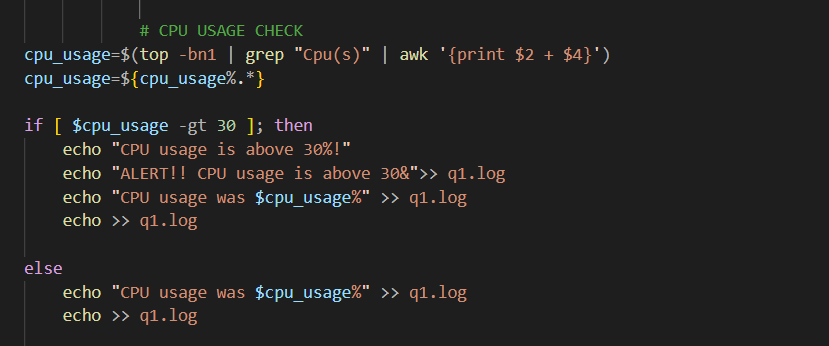
# header line always gets ignored

# int($5) >30 looks into the fifth column of the data and

# filter outs values bigger than 30%(the values are already in %)

# and lastly the entries get printed in Disk\_usage.txt

### CPU Usage Check



# top -bn1 gives the CPU usage of that instance

# grep "Cpu(s)" filters out the line with string "Cpu(s)"

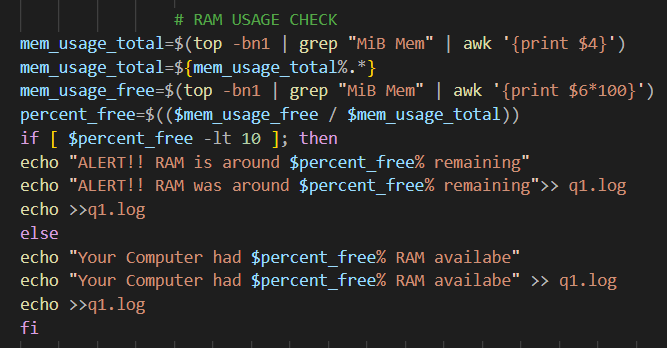
# awk '{print $2 + $4}' adds the 2nd and 4th column

# which gives us the total CPU usage

# and stores it in CPU usage

# -gt(greater than) is used to compare the value

### RAM Usage Check



# top -bn1 gives the memory usage of that instance

# grep "MiB Mem" filters out the line with the string "MiB Mem"

# awk '{print $4}' gives us the total memory usage

# and stores it in mem\_usage\_total

# -gt(greater than) is used to compare the value

# awk '{print $6\*100}' gives us the free memory

# and stores it in mem\_usage\_free

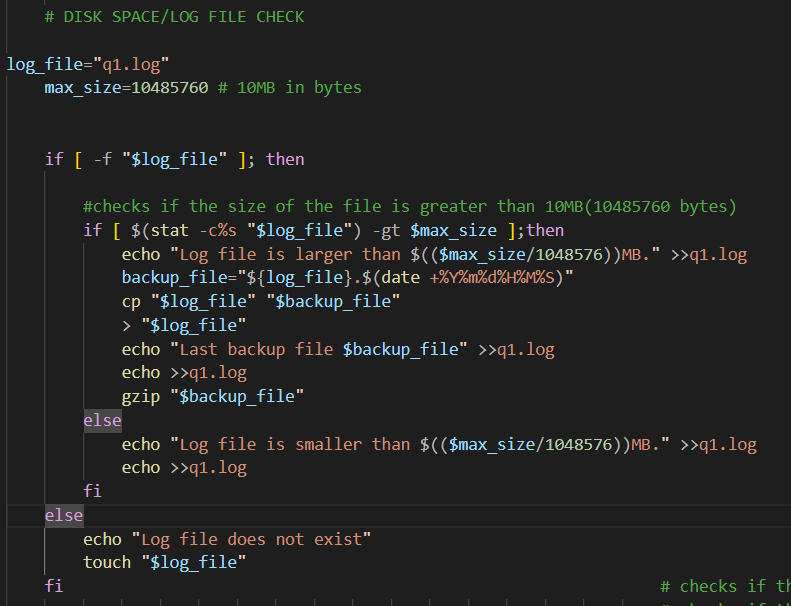
# $mem\_usage\_free / $mem\_usage\_total gives us the

# percentage of free memory

# -lt(less than) is used to compare the value

# and lastly the entries get printed in q1.log

### LogFile/Disk Space Check



# checks if the log file exists

# checks if the size of the file is greater than 10MB

# Create a backup of the log file with a timestamp

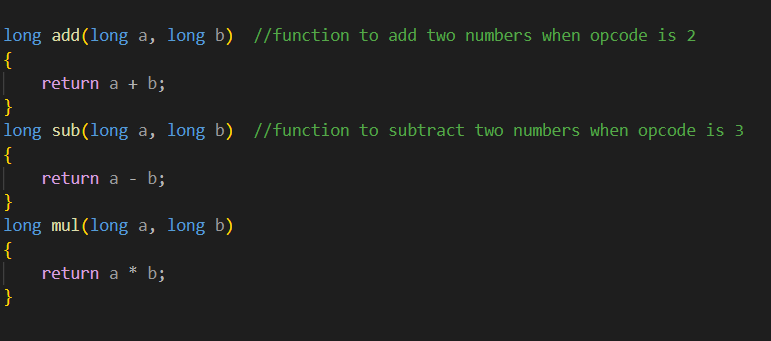
# Empty the log file

# Write the backup file name to the log file

# Compress the backup file

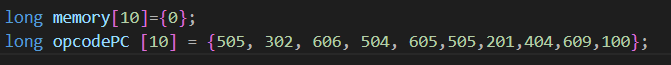
# and lastly the entries get printed in q1.log

## Question#2



// ADD, SUBTRACT and MULTIPLY functions are called when the simulator gets the

// corresponding opcodes which return the value to ACCUMULATOR.



// memory[10] will store the data when corresponding

// 100 = terminate the next operation and move to the next

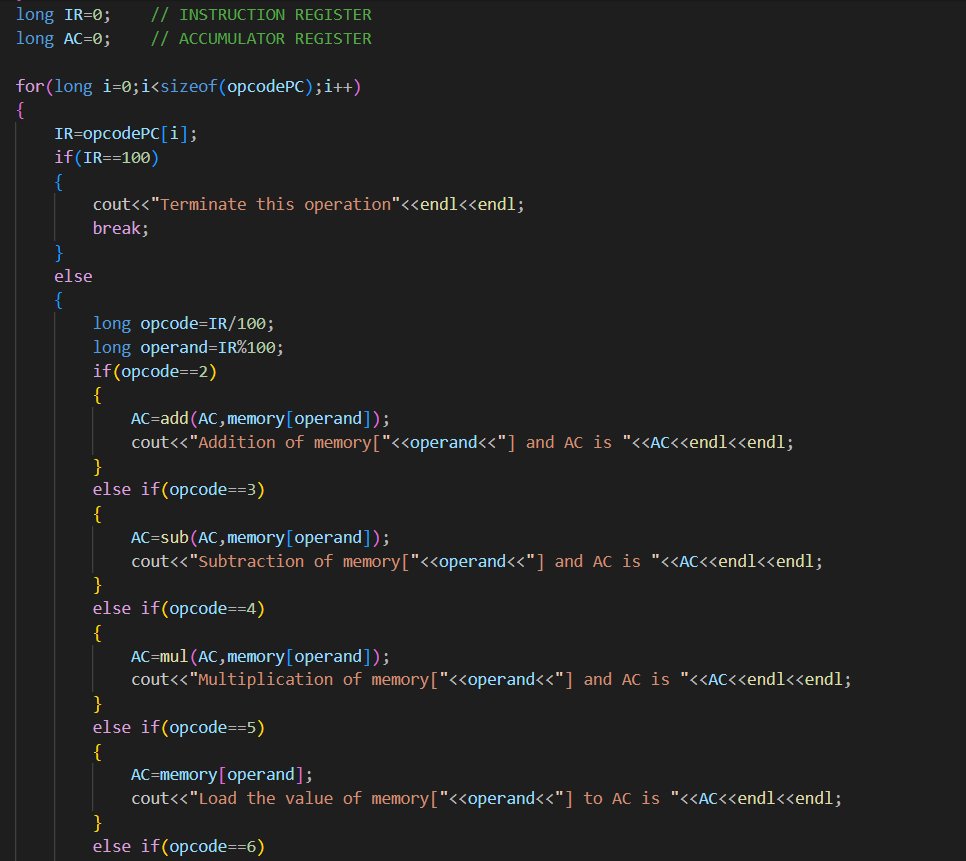
// 2 = add

// 3 = subtract

// 4 = multiply

// 5 = loads

// 6 = stores



// FOR loop runs until there is no further instruction left.

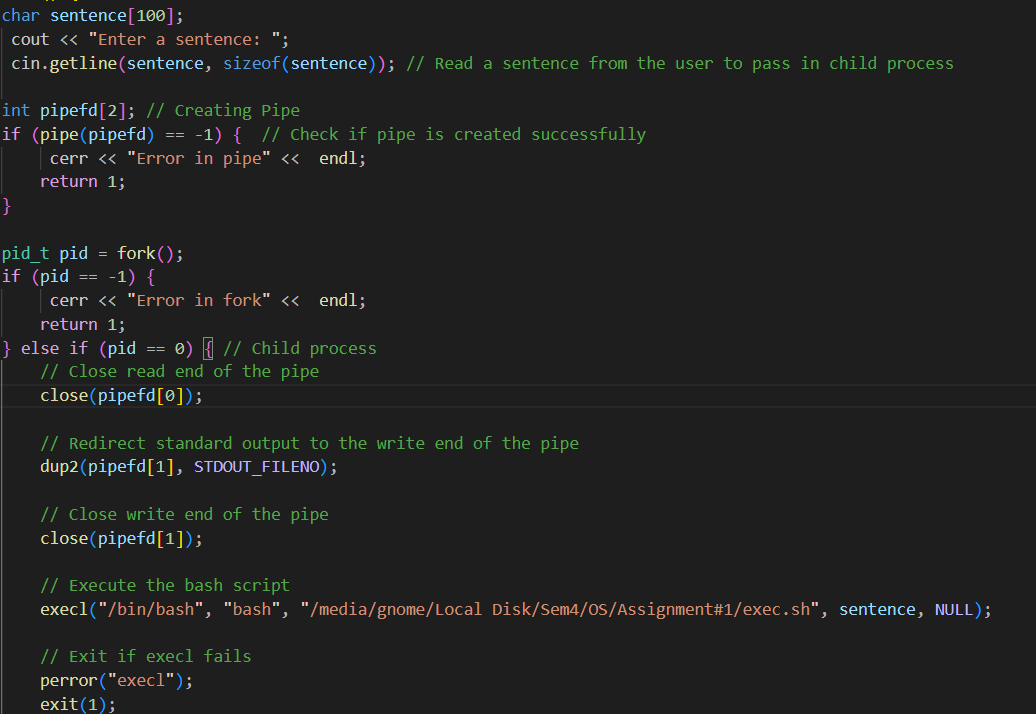
// OpCode is first loaded into IR as part of the simulation

// First integer represents the operation that will be performed

// Last two integers tell the memory location upon which the action will be performed

// IF-ELSE IF statements to do corresponding calculations

## Question#3



// cin.getline(sentence, sizeof(sentence)) Read a sentence from the user to pass in the child // process

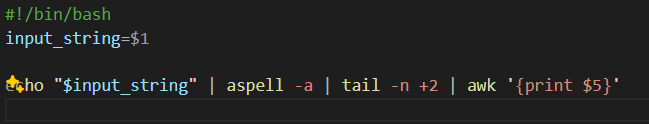
// Creating pipe and checking if it's created successfully to send the output of the child process // to parent

// We create a fork and use an IF statement to do processing in child with the help of ‘pid==0’

// We close the reading end of the pipe with ‘close(pipe[0])’ so we can write into writing end of // the pipe so parent can read it

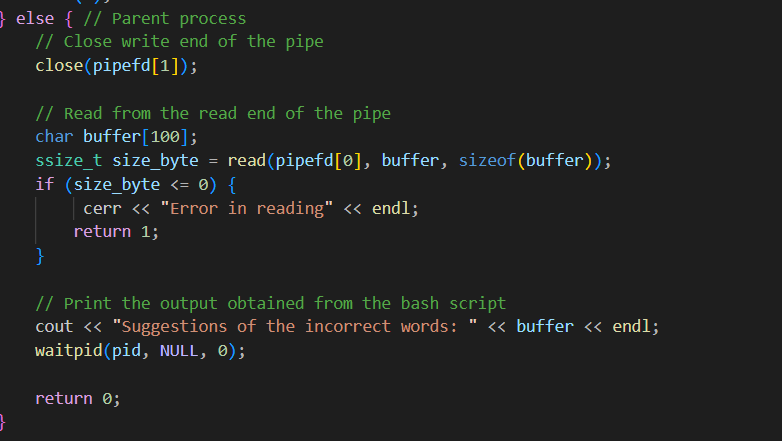
// dup2() is used to duplicate the value that the child produces in the terminal which is stored in // STDOUT\_FILENO into writing end of the pipe

// Then we call the execl() with required arguments for the script to use the spell checker using // bash



# This bash script takes the string that is in the ‘sentence’ into ‘input\_string’ and runs the # following script to check the mistakes using a program called ‘aspell’

#ouput of generated by aspell is further modified through pipes and arguments in bash



// When the child exits and the parent program starts to run again it captures the values sent // from the child and prints them into the terminal.