**Viva Preparation for Operating Systems Course LAB**

**Bahria University, Lahore Campus**

**Department of Computer Sciences**

**Operating System Lab Journals**

**Course Code: CSL-320**

**Spring 2024**

**Name: Muhammad Hammad**

**Enroll No: 03-134221-024**

**Viva Objective:**

**The objective of this viva is to assess the understanding of core concepts and practical applications covered in the Operating Systems Lab. This includes threads, process scheduling, memory management, inter-process communication, signal handling, and basic shell programming.**

**Lab Journal 01**

**Date:** 21-09-2023  
**Max Marks:** 20  
**Faculty’s Name:** Abdullah

**Objective(s):**

* To understand basic concepts of Operating System.

**Lab Tasks:**

1. **Task 1 : What is an Operating System?**
2. An operating system (OS) is the program that, after being initially loaded into the computer by a boot program, manages all of the other application programs in a computer. The application programs make use of the operating system by making requests for services through a defined application program interface (API). In addition, users can interact directly with the operating system through a user interface, such as a command-line interface (CLI) or a graphical UI (GUI).
3. **Task 2 : Which OS is being used in the Lab?**
4. Operating System used in the Lab is Windows 10 pro and UBUNBTU.
5. **Task 3 : Install VMWARE and UBUNBTU on your laptops.**
6. To install VMware and Ubuntu on your laptops, you'll first need to download VMware software from their official website and then download the Ubuntu ISO file. Once VMware is installed, you can create a new virtual machine and install Ubuntu on it by following the prompts within VMware.
7. **Task 4 : What is a Virtual Machine? Differentiate between Guest and Host OS**.
8. A Virtual Machine (VM) is a compute resource that uses software instead of a physical computer to run programs and deploy apps. One or more virtual “guest” machines run on a physical “host” machine. Each virtual machine runs its own operating system and functions separately from the other VMs, even when they are all running on the same host. This means that, for example, a virtual MacOS virtual machine can run on a physical PC.

|  |  |
| --- | --- |
| **Guest** | **Host** |
| The Guest operating system runs within the virtual machine. | The Host operating system is the operating system installed directly on the physical hardware. |
| The Guest OS operates within the virtual environment provided by the VM software. | The Host OS manages the hardware resources and provides services to the guest OS. |

1. .

**Lab Journal 02**

**Date:** 2/29/2024  
**Max Marks:** 20  
**Faculty’s Name:** Abdullah

**Objective(s):**

* To study and execute the commands in Linux.

**Lab Tasks:**

1. Execute the Date Commands and write the output.

**Syntax:** $date

## Output:

A screen shot of a computer

Description automatically generated

**Syntax:** $date +%H-%M-%S

## Output:

## A screen shot of a computer Description automatically generated

**Syntax**

$cal year

$cal month year

## Output:

A screenshot of a computer

Description automatically generated

1. Execute the mentioned LINUX Commands and generate output.

**Syntax:** $echo text

## Output:

A screenshot of a computer

Description automatically generated

**Syntax:** $banner <arguments>

## Output:

A black screen with white text

Description automatically generated

**Syntax:** $who – option’s

## Output:

## A black background with white text Description automatically generated

**Syntax:** $whoami

## Output:



**Syntax:** $clear

A black screen with white text

Description automatically generated

1. Execute the below File Commands and write the output.

**Syntax:**

$cat > filename.

The > symbol is re-directory we use cat command.

## Output:

## A screenshot of a computer Description automatically generated

.

**Syntax:**

$cat <filename.

Options –s = to neglect the warning /error message.

## Output:

A screen shot of a computer

Description automatically generated

**Syntax:** $cat –n filename

## Output:

A screenshot of a computer

Description automatically generated

**Syntax**

$cp source filename destination filename

## Output:

A screenshot of a computer

Description automatically generated

A screen shot of a computer

Description automatically generated

After copying data and running code:

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**Syntax:** $ mv source filename destination filename

## Output:

A screenshot of a computer

Description automatically generated

A screen shot of a computer

Description automatically generated

After moving:

## A screenshot of a computer Description automatically generated

A screenshot of a computer

Description automatically generated

**Syntax:** $rm filename

## Output:

A screenshot of a computer

Description automatically generated

**Syntax:** $wc filename

## Output:

A screenshot of a computer

Description automatically generated

1. Execute FILTERS AND PIPES commands and write the output.

**Syntax:** $head filename

## Output:

A screenshot of a computer

Description automatically generated

**Syntax:** $tail filename

## Output:

## A screenshot of a computer Description automatically generated

**Syntax:** $sort filename

## Output:

## A black and white text Description automatically generated

**Syntax:** echo 1+1**|**bc

## Output:

A screenshot of a computer

Description automatically generated

**Syntax:** $tr “[a-z]” “[A-Z]”

## Output:

## A screenshot of a computer Description automatically generated

**Lab Journal 03**

**Date:** 03-07-2024  
**Max Marks:** 20  
**Faculty’s Name:** Abdullah

**Objective(s):**

* Understanding of Shell Programming.
* Understanding of variables, loops, operators.

**Lab Tasks:**

1. Write the output of programs for LINUX variables.

#!/bin/bash  
 #Variable Assignment & Accessing   
 echo "Variable Name : "  
 Name="Operating System"

echo $Name

**Output:**

A screenshot of a computer

Description automatically generated

#!/bin/bash   
 #Input from user   
 echo "Enter your name"

read NAME

echo "Enter your age"

read AGE  
 echo "Enter your enrollment"  
 read ENROLLMENT  
 echo "Hello $NAME, Your age is : $AGE Your enrollment is : $ENROLLMENT**"**

page6image43302912

**OUTPUT**

A screenshot of a computer

Description automatically generated

**iii)** #!/bin/bash #readonly Variables

echo "Readonly Variables"

Name=”David”

readonly Name

Name=’John’

**OUTPUT**

A screen shot of a computer

Description automatically generated

1. Write a program to calculate the addition, subtraction, multiplication, and division of numbers.

**Program:**

echo "Enter First Number : "

read var1

echo "Enter Second Number : "

read var2

echo "Enter 1 For addition,Enter 2 For Subtraction,Enter 3 For Division,Enter 4 For Multiplication"

read x

if [[ x -eq 1 ]]; then

echo "Sum of $var1 and $var2 is $((var1 + var2))"

elif [[ x -eq 2 ]]; then

echo "Sub of $var1 and $var2 is $((var1 - var2))"

elif [[ x -eq 3 ]]; then

echo "Division of $var1 and $var2 is $((var1 / var2))"

elif [[ x -eq 4 ]]; then

echo "Multiplication of $var1 and $var2 is $((var1 \* var2))"

else

echo "Multiplication of $var1 and $var2 is $((var1 \* var2))"

fi

**Output:**

A black screen with white text

Description automatically generated

1. 3.1 Write a program that compares two numbers if a is greater than b it displays “a is greater than b”, otherwise it displays that ‘a is not equal to b’.

echo "Enter the value of a:"

read a

echo "Enter the value of b:"

read b

if [ "$a" -gt "$b" ]; then

echo "a is greater"

else

echo "a is not equal to b"

fi

**Output:**

A screenshot of a computer

Description automatically generated

1. 3.2 Write a program that compares two numbers to check whether the numbers are equal, a is greater than b or a is less than b.

**Program:**

echo "Enter the value of a:"

read a

echo "Enter the value of b:"

read b

if [ $a -gt $b ]; then

echo "a is greater than b"

elif [ $a -eq $b ]; then

echo "a is equal to b"

else

echo "a is lesser than b"

fi

**Output:**

A black screen with white text

Description automatically generated

1. Write a program using “case” that inputs a fruit from the user and displays “Apple pie” on the input of apple, “I like banana” on the input of banana and “New Zealand famous for kiwi” on the input of kiwi.

**Program:**

echo "Enter the case:"

read a

if [ $a = apple ]; then

echo "Apple Pie"

elif [ $a = banana ]; then

echo "I like banana"

elif [ $a = kiwi ]; then

echo "New Zeland famous for kiwi"

else

echo "Wrong Input"

fi

**Output:**

A screenshot of a computer

Description automatically generated

**Lab Journal 04**

**Date:** 12-10-2023  
**Max Marks:** 20  
**Faculty’s Name:** Abdullah

**Objective(s):**

* To study loops, arrays, strings, file testing, positional parameters.

**Lab Tasks:**

1. Write a program to display the numbers from 10 to 20 in reverse order using for loop.

**Code**

for var in 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

do

echo $var

done

**Output**

A screenshot of a computer

Description automatically generated

1. Write the output of the programs of Array.

#!/bin/bash

#Arrays through positional parameters

set apples bananas oranges peach

echo $1

echo $3

echo $\*

echo $#

echo $@

**OUTPUT**

A black screen with white text

Description automatically generated

1. Write the output of the File Testing Program.

#!/bin/bash

File=”Give the path of file here”

if [ -s $File ]

then

echo “File’s length is not zero”

else

echo “File’s length is zero”

fi

if [ -r $File ]

then

echo “It is readable”

else

echo “It is not readable”

fi

**OUTPUT**A screen shot of a computer program

Description automatically generated

1. Write a shell program to compare the two strings, whether the strings are equal or not.

**Code:**

echo enter the first string

read str1

echo enter the second string

read str2

if [[ $str1 = $str2 ]]; then

echo "Both String 1 and String 2 is equal"

else

echo "String 1 and String 2 are not equal"

fi

**Output:**

A black screen with white text

Description automatically generated

**Lab Journal 05**

**Date:** 29-10-2023  
**Max Marks:** 20  
**Faculty’s Name:** Abdullah

**Objective(s):**

* To write a program to create a process in LINUX.
* To understand exec process.
* To create child with sleep and wait command.
* To understand getpid() and getppid().

**Lab Tasks:**

1. Write the program for process creation using fork command.

**Program**#include<iostream>

#include <unistd.h>

using namespace std;

int main()

{

int pid = fork();

if(pid<0)

{

cout<<"Error" <<endl; //will print error

}

else if(pid==0)

{

cout<<"Child procces";

//will execute child procces

}

else

{

cout<<"Parent procces";

//will execute parent procces

}

return 0;

}

**Program Execution   
Task 1:** Write the program for process creation using fork command.

**STEP 1:** Start the program.

**STEP 2:** Declare pid as integer.

**STEP 3:** Create the process using Fork command.

**STEP 4:** Check pid is less than 0 then print error else if pid is equal to 0 then execute command else parent process wait for child process.

**STEP 5:** Exit the process.

**OUTPUT**

A screenshot of a computer

Description automatically generated

1. Write a program illustrating the sleep command during process creation.

**Program**

#include<iostream>

#include <unistd.h>

#include <sys/wait.h>

using namespace std;

int main()

{

int pid = fork();

if(pid<0)

{

cout<<"Error" <<endl; //will print error

}

else if(pid==0)

{

cout<<"Child procces " <<endl;

cout<<"Child pid = " <<getpid() << " Corresponsdin pid = " << getppid()<<endl;

sleep(5);

cout<<"Child pid after child woke up = " <<getpid() <<endl;

cout<<"Parent pid = " <<getppid() <<endl;

}

else

{

cout<<"Parent procces"<<endl;

sleep(10);

cout<<"Parent pid after parent woke up = " <<getppid()<<endl;

}

return 0;

}

**Program Execution**

**STEP 1:** Start the execution and create a process using fork( ) command.  
**STEP 2:** Make the parent process to sleep for 10 seconds.   
**STEP 3:** In the child process print it pid and it corresponding pid.

**STEP 4:** Make the child process to sleep for 5 seconds.  
**STEP 5:** Again print it pid and it parent pid.  
**STEP 6:** After making the sleep for the parent process for 10 seconds print it pid.  
**STEP 7:** Stop the execution.

**OUTPUT**

**A screenshot of a computer screen

Description automatically generated**

1. Write a program illustrating the wait command during process creation.

**Program**

#include<iostream>

#include <unistd.h>

#include <sys/wait.h>

using namespace std;

int main()

{

int pid = fork();

int i = 5;

if(pid<0)

{

cout<<"Error" <<endl; //will print error

}

else if(pid==0)

{

cout<<"Child procces " <<endl;

//will execute child procces

cout<<"Value of I is : " << i <<endl ;

}

else

{

cout<<"Parent procces";

//will execute parent procces

wait(NULL);

}

return 0;

}

**Program Execution**

**STEP 1**: Start the execution  
**STEP 2**: Create process using fork and assign it to a variable  
**STEP 3**: Check for the condition pid is equal to 0  
**STEP 4**: If it is true print the value of i and terminate the child process   
**STEP 5**: If it is not a parent process has to wait until the child terminate   
**STEP 6**: Stop the execution

**OUTPUT**

**A screenshot of a computer

Description automatically generated**

1. Write a program in C to create two processes Parent and Child through fork.

**Program**

#include<iostream>

#include <unistd.h>

#include <sys/wait.h>

using namespace std;

int main()

{

int pid = fork();

int num;

if(pid<0)

{

cout<<"Error" <<endl; //will print error

}

else if(pid==0)

{

cout<<"Child procces " <<endl;

cout<<"Child pid = " <<getpid() << " Parent pid = " << getppid() <<endl;

cout<<"Enter a number for Child procces : ";

cin>>num;

cout << "Square of " << num << " is " << num \* num << endl;

}

else

{

cout<<"Parent procces"<<endl;

cout<<"Parent pid = " <<getpid() << " Child pid = " <<getppid() <<endl;

cin.ignore();

cout<<"Enter a number for parent procces : ";

cin>>num;

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

{

cout<< num << " x " << i << " = " << num\*i <<endl;

}

wait(NULL);

}

return 0;

}

**Program Execution**

**STEP 1:** Start the execution and create a process using fork( ) command.  
**STEP 2:** Display Parent Id and Child Id in Parent Process   
**STEP 3:** Display Parent Id and Child Id in Child Process

**STEP 4:** Take Input For Parent Process and make its table upto 10 and display table.

**STEP 5:** Take Input For Childe Process and Square it and display   
**STEP 6:** Stop the execution.

**OUTPUT**

**A screenshot of a computer

Description automatically generated**

**Lab Journal 06**

**Date:** 03-28-2024  
**Max Marks:** 20  
**Faculty’s Name:** Abdullah

**Objective(s):**

* To understand how the processes will cooperate by communicating with each other using the approach called as shared memory approach.

**Lab Tasks:**

**Task 01:** Write the steps for sharing a common memory segment

## Answer:

**Creating a Shared Memory Segment**

**Server:**

It should start before any client and should do thee following steps.

1. Ask for a shared memory with a memory key and memorize the returned shared memory ID. This is performed by system call shmget( ).
2. Attach shared memory to the server's address space with system call shmat( ).
3. Initialize the shared memory, if necessary.
4. Do whatever you want in shared memory (write something) and wait for all clients' completion.
5. Detach the shared memory with system call shmdt( ).
6. Remove the shared memory with system call shmctl( ).

**Client:**

1. Ask for a shared memory with the same memory key and memorize the returned shared memory ID. Do not write the flag IPC\_CREAT in the shmget() call from client.
2. Attach this shared memory to the client's address space with system call shmget().
3. Use the Shared memory (Read what was written by server).
4. Detach all shared memory segments, if necessary.
5. **Requesting for a Shared Memory Segment - shmget( ):**

Shmget( ) returns value

* 0 successfully get the requested shared memory < 0 failed to get the requested shared memory

**Key:**

A key is simply an integer of type key\_t; however, you should not use int or long, since the length of a key is system dependent.

There are three different ways of using keys, namely:

1. a specific integer value (e.g., 123456)
2. a key generated with function ftok( )
3. a uniquely generated key using IPC\_PRIVATE (i.e., a private key).

**Task 02:** Write the functions required for creating, attaching, detaching and removing the memory segment.

## Answer:

**2) Attaching a Shared Memory Segment to an Address Space - shmat( ) :**

Suppose process 1, a server, uses shmget( ) to request a shared memory segment successfully. That shared memory segment exists somewhere in the memory, but is not yet part of the address space of process

Similarly, if process 2 requests the same shared memory segment with the same key value, process 2 will be granted the right to use the shared memory segment; but it is not yet part of the address space of process

So to make a requested shared memory segment part of the address space of a process, we use shmat( ).

A white background with black and white clouds

Description automatically generatedA black background with white text

Description automatically generated

System call shmat( ) accepts a shared memory ID, shm\_id, and attaches the indicated shared memory to the program's address space. The returned value is a pointer of type (void \*) to the attached shared memory. Thus, casting is usually necessary. If this call is unsuccessful, the return value is -1. Normally, the second parameter is NULL. If the flag is SHM\_RDONLY, this shared memory is attached as a read-only memory; otherwise, it is readable and writable.

**3) Writing/Reading the Shared Memory Segment:**

The pointer shm\_ptr returned returned by shmat() call points to the start of shared memory. To write or read from the shared memory we use this pointer.

**4) Detaching a Shared Memory Segment -shmdt( ):**

System call shmdt( ) is used to detach a shared memory.

After a shared memory is detached, it cannot be used. However, it is still there and can be re-attached back to a process's address space, perhaps at a different address.

where shm\_ptr is the pointer to the shared memory. This pointer is returned by shmat( ) when the shared memory is attached.

## 

**5) Removing a Shared Memory Segment -shmctl( ):**

To remove a shared memory, use shmctl( ).

where shm\_id is the shared memory ID. IPC\_RMID indicates this is a remove operation. This step is performed only from the sever side.

**Task 03:** Write a program in C to perform communication between parent and child through shared memory.

• Parent should be the server and child should be the client.

• Differentiate between parent and child through conditional statements after the fork system call.

• Parent:

Should create the shared memory and write Capital English “HELLO” in it, through those 5 steps taught.

• Child:

Should access that memory and read whatever is written by parent in it.

## Code:

#include <stdio.h>

#include <stdlib.h>

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <unistd.h>

#define SHMSIZE 6

int main() {

int shmid;

key\_t key;

char \*shm, \*s;

pid\_t pid;

key = ftok(".", 'a');

if ((shmid = shmget(key, SHMSIZE, IPC\_CREAT | 0666)) < 0) {

perror("shmget");

exit(1);

}

if ((shm = shmat(shmid, NULL, 0)) == (char \*) -1) {

perror("shmat");

exit(1);

}

pid = fork();

if (pid < 0) {

perror("fork");

exit(1);

}

if (pid == 0) {

printf("Child process (client) is reading from shared memory...\n");

printf("Child process read: %s\n", shm);

if (shmdt(shm) == -1) {

perror("shmdt");

exit(1);

}

} else {

printf("Parent process (server) is writing to shared memory...\n");

s = shm;

\*s++ = 'H';

\*s++ = 'E';

\*s++ = 'L';

\*s++ = 'L';

\*s++ = 'O';

\*s = '\0';

wait(NULL);

if (shmdt(shm) == -1) {

perror("shmdt");

exit(1);

}

if (shmctl(shmid, IPC\_RMID, NULL) == -1) {

perror("shmctl");

exit(1);

}

}

return 0;

}

## Output:

A black screen with white text

Description automatically generated

**Task 04:** Write a program that creates a shared memory segment and waits until three other separate processes writes something into that shared memory segment after which it prints what is written in shared memory.

**Answer :**

*#include <sys/types.h>*

*#include <sys/ipc.h>*

*#include <sys/shm.h>*

*#include <stdio.h>*

*#include <stdlib.h>*

*#include <unistd.h>*

*#define SHMSIZE 27*

*int main(){*

*char c;*

*int shmid;*

*char \*shm, \*s;*

*key\_t key;*

*key = 5678;*

*//Create Segment*

*if ((shmid= shmget(key, SHMSIZE, IPC\_CREAT | 0666))<0){*

*perror("shmget");*

*exit(1);*

*}*

*//Now attach the segment to our dataspace*

*shm=shmat(shmid, NULL,0);*

*//Now put 1 in first place*

*s=shm;*

*\*s++='F';*

*\*s++='I';*

*\*s++='R';*

*\*s++='S';*

*\*s++='T';*

*printf("Process 1 : I have put the message %s\n",shm);*

*//Finally, we waitt until the other process changes the first character of memory*

*while(\*shm !='2')*

*{*

*s=shm;*

*s++;*

*\*s++=' ';*

*\*s++='h';*

*\*s++='e';*

*\*s++='l';*

*\*s++='l';*

*\*s++='o';*

*//Now put 2 in first place*

*\*shm='2';*

*sleep(1);*

*printf("Process 1: Process 2 has put the messages %s\n",shm);*

*}*

*while(\*shm !='3')*

*{*

*s=shm;*

*s++;*

*\*s++=' ';*

*\*s++='m';*

*\*s++='e';*

*\*s++='m';*

*\*s++='o';*

*\*s++='r';*

*\*s++='y';*

*//Now put 3 in first place*

*\*shm='3';*

*sleep(1);*

*printf("Process 1: Process 3 has put the message %s\n",shm);*

*printf("Process 1: I am quiting\n");*

*}*

*shmdt(shm);*

*shmctl(shmid, IPC\_RMID,NULL);*

*return 0;*

*}*

**OUTPUT**

A screenshot of a computer

Description automatically generated

**Lab Journal 07**

**Date:** 05/02/2024  
**Max Marks:** 20  
**Faculty’s Name:** Abdullah

**Objective(s):**

* To write a C program to implement the CPU scheduling algorithm for First Come First Serve.
* To write a C program to implement the CPU scheduling algorithm for Shortest Job First.

**Lab Tasks:**

1. Calculate the Average Time using FCFS Algorithm.
2. Write the program for First Come First Serve scheduling algorithm.

Task 1 and 2:

**Program**

#include<iostream>

using namespace std;

void findWaitingTime( int n,int bt[], int wt[])

{

wt[0] = 0;

for (int i = 1; i < n; i++)

{

wt[i] = bt[i - 1] + wt[i - 1];

}

}

void findTurnAroundTime( int n,int bt[], int wt[], int tat[])

{

for (int i = 0; i < n; i++)

{

tat[i] = bt[i] + wt[i];

}

}

void findavgTime(int p[], int n, int bt[])

{

int wt[3], tat[3], twt = 0, ttat = 0;

findWaitingTime( n, bt, wt); //waiting time

findTurnAroundTime(n, bt, wt, tat); //turn around time

cout << "Processes " << " Burst time " << " Waiting time " << " Turn around time\n";

for (int i = 0; i<n; i++)

{

twt = twt + wt[i];

ttat = ttat + tat[i];

cout << " " << i + 1 << "\t\t" << bt[i] << "\t "<< wt[i] << "\t\t " << tat[i] << endl;

}

cout << "Average waiting time = " << (double)twt / (double)n << "\nAverage turn around time = " << (double)ttat / (double)n;

}

int main()

{

int p[] = { 1, 2, 3 };

int n = sizeof p / sizeof p[0];

int bt[] = { 23, 3, 4};

findavgTime(p, n, bt);

system("pause");

return 0;

}

**OUTPUT**

A screenshot of a computer screen

Description automatically generated

1. Calculate the Average Time using SJF Algorithm.
2. Write the program for Shortest Job First scheduling algorithm.\

Task 3 and 4

|  |  |
| --- | --- |
| **Program**  #include<iostream>  using namespace std;  void findWaitingTime( int n,int bt[], int wt[])  {  wt[0] = 0;  for (int i = 1; i < n; i++)  {  wt[i] = bt[i - 1] + wt[i - 1];  }  }  void findTurnAroundTime( int n,int bt[], int wt[], int tat[])  {  for (int i = 0; i < n; i++)  {  tat[i] = bt[i] + wt[i];  }  }  void makesjfarr(int p[],int bt[], int n)  {  for (int i = 0; i < n; i++)  {  for (int j = i + 1; j <n; j++)  {  if (bt[i] > bt[j])  {  swap(p[i], p[j]);  swap(bt[i], bt[j]);  }  }    }  }  void findavgTime(int p[], int n, int bt[])  {  int wt[4], tat[4], twt = 0, ttat = 0;  makesjfarr(p, bt, n);  findWaitingTime( n, bt, wt); //waiting time  findTurnAroundTime(n, bt, wt, tat); //turn around time  cout << "Processes " << " Burst time " << " Waiting time " << " Turn around time\n";  for (int i = 0; i<n; i++)  {  twt = twt + wt[i];  ttat = ttat + tat[i];  cout << " p" << p[i]<< "\t\t" << bt[i] << "\t "<< wt[i] << "\t\t " << tat[i] << endl;  }  cout << "Average waiting time = " << (double)twt / (double)n << "\nAverage turn around time = " << (double)ttat / (double)n << "\nThroughput time = " << (double)n / (double)tat[3];  }  int main()  {  int p[] = { 1, 2, 3 ,4};  int n = sizeof p / sizeof p[0];  int bt[] = { 6,8,7,3};  findavgTime(p, n, bt);  cout << endl;  system("pause");  return 0;  } |  |

**OUTPUT**A black screen with white text

Description automatically generated

**Lab Journal 08**

**Date:** 11 Nov 2023  
**Max Marks:** 20  
**Faculty’s Name:** Abdullah

**Objective(s):**

* To write a C program to implement CPU scheduling algorithm for Priority Scheduling and Shortest Remaining Time First.

**Lab Tasks:**

1. Write a C program to implement Priority Scheduling algorithm.

Code:

#include <iostream>

#include <algorithm>

using namespace std;

struct Process {

int id;

int burst\_time;

int priority;

int waiting\_time;

int total\_time;

};

bool comparePriority(const Process &a, const Process &b) {

return a.priority < b.priority;

}

int main() {

int n;

cout << "Enter the number of processes: ";

cin >> n;

Process processes[n];

// Input process details

for (int i = 0; i < n; i++) {

cout << "Enter Process ID, Burst Time, and Priority for process " << i + 1 << ": ";

cin >> processes[i].id >> processes[i].burst\_time >> processes[i].priority;

processes[i].waiting\_time = 0;

processes[i].total\_time = 0;

}

// Sort processes based on priority

sort(processes, processes + n, comparePriority);

// Calculate waiting time and total time

processes[0].waiting\_time = 0;

processes[0].total\_time = processes[0].burst\_time;

for (int i = 1; i < n; i++) {

processes[i].waiting\_time = processes[i - 1].total\_time;

processes[i].total\_time = processes[i].waiting\_time + processes[i].burst\_time;

}

// Calculate average waiting time and average turnaround time

double avg\_waiting\_time = 0, avg\_turnaround\_time = 0;

for (int i = 0; i < n; i++) {

avg\_waiting\_time += processes[i].waiting\_time;

avg\_turnaround\_time += processes[i].total\_time;

}

avg\_waiting\_time /= n;

avg\_turnaround\_time /= n;

// Display process details

cout << "\nId \t Burst Time \t Priority \t Waiting Time \t Total Time\n";

for (int i = 0; i < n; i++) {

cout << processes[i].id << "\t" << processes[i].burst\_time << "\t\t" << processes[i].priority << "\t\t"

<< processes[i].waiting\_time << "\t\t" << processes[i].total\_time << endl;

}

// Display average waiting time and average turnaround time

cout << "\nAverage Waiting Time: " << avg\_waiting\_time << endl;

cout << "Average Turnaround Time: " << avg\_turnaround\_time << endl;

return 0;

}

Output:

A screenshot of a computer screen

Description automatically generated

1. Write the output of a C program for Shortest Remaining Time First.

Code:

#include <iostream>

using namespace std;

int main() {

int arrival\_time[10], burst\_time[10], temp[10];

int count = 0, limit;

// Input the number of processes

cout << "Enter the Total Number of Processes: ";

cin >> limit;

// Input arrival time and burst time for each process

cout << "Enter Details of " << limit << " Processes\n";

for (int i = 0; i < limit; i++) {

cout << "Enter Arrival Time for Process " << i+1 << ": ";

cin >> arrival\_time[i];

cout << "Enter Burst Time for Process " << i+1 << ": ";

cin >> burst\_time[i];

temp[i] = burst\_time[i]; // Store burst time in temporary array

}

// Simulate the scheduling algorithm

int time = 0;

while (count != limit) {

int smallest = -1; // Index of the process with the smallest burst time

for (int i = 0; i < limit; i++) {

if (arrival\_time[i] <= time && burst\_time[i] > 0) {

if (smallest == -1 || burst\_time[i] < burst\_time[smallest]) {

smallest = i;

}

}

}

// If no process is ready to execute, move time forward

if (smallest == -1) {

time++;

continue;

}

burst\_time[smallest]--; // Execute the process for one time unit

time++; // Increment time

if (burst\_time[smallest] == 0) { // If process completes execution

count++; // Increment completed process count

int end = time;

int wait = end - arrival\_time[smallest] - temp[smallest];

int turnaround = end - arrival\_time[smallest];

cout << "Process " << smallest+1 << " Completed.\n";

cout << "Wait Time: " << wait << ", Turnaround Time: " << turnaround << endl;

}

}

// Calculate and output average waiting time and turnaround time

double total\_wait\_time = 0, total\_turnaround\_time = 0;

for (int i = 0; i < limit; i++) {

int end = arrival\_time[i] + temp[i];

total\_wait\_time += end - arrival\_time[i] - temp[i];

total\_turnaround\_time += end - arrival\_time[i];

}

double average\_waiting\_time = total\_wait\_time / limit;

double average\_turnaround\_time = total\_turnaround\_time / limit;

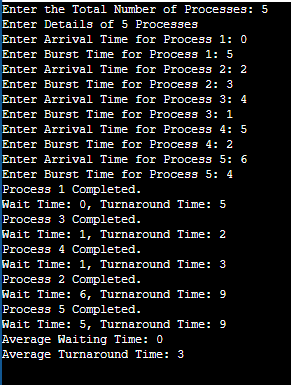
cout << "Average Waiting Time: " << average\_waiting\_time << endl;

cout << "Average Turnaround Time: " << average\_turnaround\_time << endl;

return 0;

}

Output:



**Lab Journal 09**

**Date:** 5/16/24  
**Max Marks:** 20  
**Faculty’s Name:** Abdullah

**Objective(s):**

* To write a C program to implement CPU scheduling algorithm for Round Robin.

**Lab Tasks:**

1. Calculate the Average Time using Round Robin. Draw the GANTT Chart.

Task 01: Calculate the Average Time using Round Robin. Draw the GANTT Chart. TQ=2

Process No Arrival Time (AT) Burst Time (BT) Complete Time (CT) Turnaround Time (CT-TAT) Waiting Time (TAT-BT) Response Time

P1 0 10 20 20 10 0

P2 1 4 10 9 10 1

P3 2 5 17 15 10 2

P4 3 3 13 10 10 3

Ready Queue:

Process P1:

AT: 0, BT: 10

Remaining BT: 10

Process P2:

AT: 1, BT: 4

Remaining BT: 4

Process P3:

AT: 2, BT: 5

Remaining BT: 5

Process P4:

AT: 3, BT: 3

Remaining BT: 3

Gantt Chart:

| P1 | P1 | P2 | P2 | P3 | P3 | P4 | P4 | P1 | P3 | P1 |

 Average Turnaround Time: 13.5

 Average Waiting Time: 8

2. Step 2: Calculate Turnaround Time (TAT), and Waiting Time (WT):

Average Turnaround Time (TAT):

(TAT\_P1 + TAT\_P2 + TAT\_P3 + TAT\_P4) / 4

Average Waiting Time (WT):

(WT\_P1 + WT\_P2 + WT\_P3 + WT\_P4) / 4

1. Write the output for Round Robin Scheduling Algorithm.

Code:

#include <iostream>

using namespace std;

int main() {

int count, j, n, time, remain, flag = 0, time\_quantum;

int wait\_time = 0, turnaround\_time = 0, at[10], bt[10], rt[10];

cout << "Enter Total Process: \t";

cin >> n;

remain = n;

for (count = 0; count < n; count++) {

cout << "Enter Arrival Time and Burst Time for Process Process Number " << count + 1 << ": ";

cin >> at[count] >> bt[count];

rt[count] = bt[count];

}

cout << "Enter Time Quantum: \t";

cin >> time\_quantum;

cout << "\n\nProcess\t | Turnaround Time | Waiting Time\n\n";

for (time = 0, count = 0; remain != 0;) {

if (rt[count] <= time\_quantum && rt[count] > 0) {

time += rt[count];

rt[count] = 0;

flag = 1;

}

else if (rt[count] > 0) {

rt[count] -= time\_quantum;

time += time\_quantum;

}

if (rt[count] == 0 && flag == 1) {

remain--;

cout << "P[" << count + 1 << "]\t\t" << time - at[count] << "\t\t" << time - at[count] - bt[count] << endl;

wait\_time += time - at[count] - bt[count];

turnaround\_time += time - at[count];

flag = 0;

}

count = (count + 1) % n; // Move to the next process in a circular manner

}

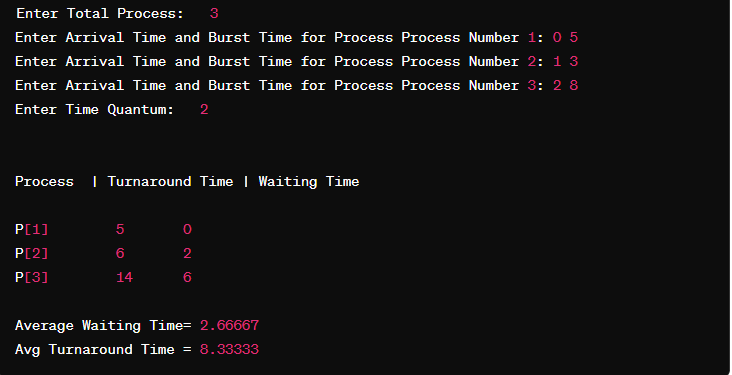
cout << "\nAverage Waiting Time= " << wait\_time \* 1.0 / n << endl;

cout << "Avg Turnaround Time = " << turnaround\_time \* 1.0 / n << endl;

return 0;

}

OUTPUT:



**Lab Journal 10**

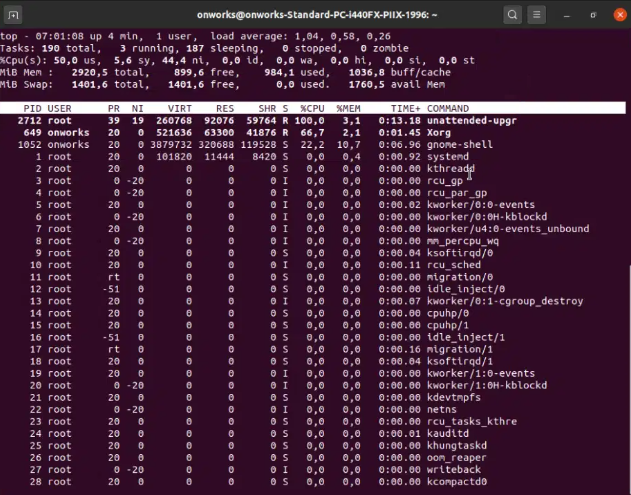
**Date:** 5/23/2024  
**Max Marks:** 20  
**Faculty’s Name:** Abdullah

**Objective(s):**

* To study about Signal Handling. Use top, ps, and Kill commands.

**Lab Tasks:**

1. Write the output for top and ps. Differentiate between the both terms.



A screenshot of a computer program

Description automatically generated

1. Write the output for the use of aux with ps for the firefox program.

A screenshot of a computer

Description automatically generated

1. Write the output of program for Kill Signal for firefox.

A computer screen with white text

Description automatically generated

1. 4.1 Write a C program with an infinite loop and a custom signal handler to handle the interrupt signal (Ctrl+C).

#include<stdio.h>

#include<signal.h>

// Ctrl-C at keyboard

void handle\_sigint(int sig){

printf("Caught signal %d\n", sig);

}

int main( ){

signal(SIGINT, handle\_sigint);

while (1){

printf("hello world\n");

sleep(1);

}

return 0;

}

**Output**

## A screen shot of a computer screen Description automatically generated

1. 4.2 Write a program in C with an infinite loop and a custom signal handler to handle at least kill -15 (SIGTERM) and kill -9 (SIGKILL). Send both these signals (kill -9 and -15) using your running process’s PID.

#include <stdio.h>

#include <stdlib.h>

#include <signal.h>

#include <unistd.h>

void signal\_handler(int signum) {

switch (signum) {

case SIGTERM:

printf("Received SIGTERM signal (kill -15).\n");

exit(EXIT\_SUCCESS);

break;

case SIGINT:

printf("Received SIGINT signal (Ctrl+C).\n");

exit(EXIT\_SUCCESS);

break;

default:

printf("Received unexpected signal.\n");

}

}

int main() {

if (signal(SIGTERM, signal\_handler) == SIG\_ERR) {

perror("Unable to catch SIGTERM");

exit(EXIT\_FAILURE);

}

if (signal(SIGINT, signal\_handler) == SIG\_ERR) {

perror("Unable to catch SIGINT");

exit(EXIT\_FAILURE);

}

// Infinite loop

while (1) {

printf("Hello World\n");

sleep(1);

}

return 0;

}

## Output:

**Sending Signal:**

## A screen shot of a computer Description automatically generated

**Lab Journal 11**

**Date:** 05/30/24  
**Max Marks:** 20  
**Faculty’s Name:** Abdullah

**Objective(s):**

* Understanding of threads, creation of threads, passing arguments to threads and to joining threads.

**Lab Tasks:**

1. Create a simple program for thread creation and termination. The created thread should display “I am child thread”.

**Program**

#include<iostream>

#include <thread>

using namespace std;

void child\_thread()

{

cout << "I am child thread\n";

}

int main()

{

thread t(child\_thread);

t.join();

cout << "Termination\n";

system("pause");

return 0;

}

**OUTPUT**

## A screenshot of a computer Description automatically generated

1. Write a program that creates a number of threads and each thread should print “Hello World!” along with its number passed as argument in a thread.

**Program**

#include<iostream>

#include <thread>

using namespace std;

void display()

{

cout << "Hello World!\n";

}

int main()

{

thread t0(display);

thread t1(display);

thread t2(display);

thread t3(display);

thread t4(display);

t0.join();

t1.join();

t2.join();

t3.join();

t4.join();

cout << "All Threads Termination\n";

system("pause");

return 0;

}

**OUTPUT**

## A screenshot of a computer Description automatically generated

1. Write the output of the program for the difference between processes and threads. Check the values of Global and Local variables in threads and processes.

**Program**

#include <stdio.h>

#include <pthread.h>

#include <unistd.h>

#include <sys/types.h>

#include <wait.h>

int global\_var=10;

void \*thread\_func(void \*var);

int main(){

int local\_var=20;

pid\_t id;

pthread\_t thread1,thread2;

printf("Create two threads to see what content they share\n");

pthread\_create(&thread1,NULL,thread\_func,NULL);

pthread\_create(&thread2,NULL,thread\_func,NULL);

pthread\_join(thread1,NULL);

pthread\_join(thread2,NULL);

printf("\nAfter thread global var = %d",global\_var);

printf("\n");

printf("\nBefore Fork :\nGlobal Variable:%d \nLocal Variable:%d",global\_var,local\_var);

id=fork();

if(id==0){

printf("\nIn child: \nGlobal Variable=%d \nLocal Variable=%d",global\_var,local\_var);

local\_var=133; global\_var=100;

printf("\nChild set: Global variable=%d \nLocal Variable=%d",global\_var,local\_var);

}

else{

wait(NULL);

printf("\nIn parent: \nGlobal Variable:%d \nLocal Variable=%d",global\_var,local\_var);

}

return 0; }

void \*thread\_func(void \*var){

int local\_var=200;

printf("\nThread :\n Global Variable :%d\n Local Variable:%d ",global\_var,local\_var);

global\_var++;

printf("\nThread after incrementing global variable ::\n Global Variable :%d \n Local Variable:%d ",global\_var,local\_var);

pthread\_exit(0);

}

**OUTPUT**

A screenshot of a computer program

Description automatically generated

1. 4.1 Write a program to create two threads. One should take input from user and stores the factorial of that input. Other should take two variable base and power, calculate power.

**Code:**

#include <iostream>

#include <thread>

using namespace std;

void calculateFactorial() {

int n;

cout << "Enter a number to calculate its factorial: ";

cin >> n;

int factorial = 1;

for (int i = 1; i <= n; ++i) {

factorial \*= i;

}

cout << "Factorial of " << n << " is: " << factorial << endl;

}

void calculatePower() {

int base, power;

cout << "Enter base and power separated by space: ";

cin >> base >> power;

int result = 1;

for (int i = 0; i < power; ++i) {

result \*= base;

}

cout << base << " raised to the power " << power << " is: " << result << endl;

}

int main() {

thread t0(calculateFactorial);

t0.join();

thread t1(calculatePower);

t1.join();

cout << "All Threads Terminated" << endl;

system("pause");

return 0;

}

**Output:**

A screenshot of a computer

Description automatically generated

1. 4.2 Create ten threads. A global variable is declared sum. As each thread is created a function is called to calculate the sum of number from 1-10.

#include <iostream>

#include <thread>

using namespace std;

int Sum = 0;

int Count = 0;

void calculate()

{

int cal = 0;

Count++;

cout << "Thread Created " << Count<<endl;

for (int i = 1; i < 11; i++)

cal += i;

Sum += cal;

}

int main() {

thread t0(calculate);

t0.join();

thread t1(calculate);

t1.join();

thread t3(calculate);

t3.join();

thread t4(calculate);

t4.join();

thread t5(calculate);

t5.join();

thread t6(calculate);

t6.join();

thread t7(calculate);

t7.join();

thread t8(calculate);

t8.join();

thread t9(calculate);

t9.join();

thread t10(calculate);

t10.join();

cout << "Gobal Sum = "<<Sum<<endl<<"All Threads Terminated" << endl;

system("pause");

return 0;

}

**Output:**

A screenshot of a computer

Description automatically generated

**Lab Journal 12**

**Date:** 6/06/24  
**Max Marks:** 20  
**Faculty’s Name:** Abdullah

**Objective(s):**

* To understand the use of semaphores and how they are used in process synchronization.

**Lab Tasks:**

1. Write a program to create two processes that share a common variable using a semaphore to synchronize access.
2. Write a program to implement the Producer-Consumer problem using semaphores.
3. Write a program to implement the Reader-Writer problem using semaphores.

**Lab Journal 13**

**Date:** 6/13/24  
**Max Marks:** 20  
**Faculty’s Name:** Abdullah

**Objective(s):**

* To understand the implementation of Deadlock avoidance using Banker's Algorithm.

**Lab Tasks:**

1. Write a program to simulate Banker's Algorithm for Deadlock avoidance.
2. Write a program to detect Deadlock in a given set of processes.

**Lab Journal 14**

**Date:** 6/20/24  
**Max Marks:** 20  
**Faculty’s Name:** Abdullah

**Objective(s):**

* To understand paging and segmentation in memory management.

**Lab Tasks:**

1. Write a program to simulate Paging in memory management.
2. Write a program to simulate Segmentation in memory management.

**Lab Journal 15**

**Date:** 6/27/24  
**Max Marks:** 20  
**Faculty’s Name:** Abdullah

**Objective(s):**

* To understand file systems and how they are managed in an operating system.

**Lab Tasks:**

1. Write a program to simulate the creation and deletion of files.
2. Write a program to simulate the management of file permissions.