2. Write a shell script to read three numbers from standard input and print the minimum value and maximum.

Source Code:-

#To find minimum and maximum among three numbers

#!/bin/sh

echo "Enter three numbers:\n"

read a

read b

read c

if [ $a -gt $b -a $a -gt $c ]

then

echo "$a is greatest\n"

elif [ $b -gt $a -a $b -gt $c ]

then

echo "$b is greatest\n"

else

echo "$c is greatest\n"

fi

if [ $a -lt $b -a $a -lt $c ]

then

echo "$a is smallest"

elif [ $b -lt $a -a $b -lt $c ]

then

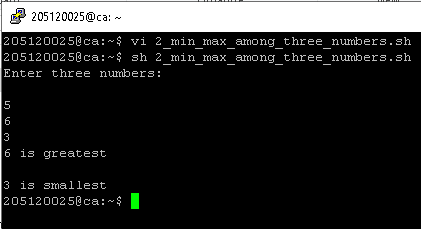
echo "$b is smallest"

else

echo "$c is smallest"

fi

Output:-



3. Write a shell script to swap two numbers without using 3rd variable.

Source Code:-

#!bin/sh

echo "Enter the first number"

read a

echo "Enter the second number"

read b

echo "a is before swapping-> $a\n"

echo "b is before swapping-> $b\n"

a=$((a+b))

b=$((a-b))

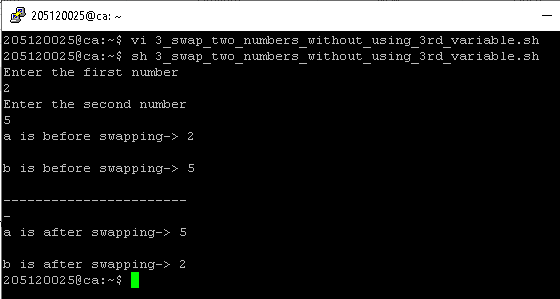
a=$((a-b))

echo "-----------------------\n-"

echo "a is after swapping-> $a\n"

echo "b is after swapping-> $b"

Output:-



4. Write a shell script to read the marks of a Student and print the grade.

Source Code:-

#!bin/sh

echo "Enter the marks of students:\n"

read m1

read m2

read m3

read m4

read m5

echo "Marks obtained:\n"

sum=$(($m1+$m2+$m3+$m4+$m5))

echo "$sum"

per=$(($sum/5))

echo "---------------------"

echo "Percentage is-> $per%"

echo "---------------------"

if [ $per -ge 60 ]

then

echo "-----------"

echo "Super Grade"

echo "-----------"

elif [ $per -ge 50 -a $per -le 59 ]

then

echo "-----------"

echo "First Grade"

echo "-----------"

elif [ $per -ge 40 -a $per -le 49 ]

then

echo "------------"

echo "Second Grade"

echo "------------"

elif [ $per -ge 30 -a $per -le 39 ]

then

echo "-----------"

echo "Third Grade"

echo "-----------"

else

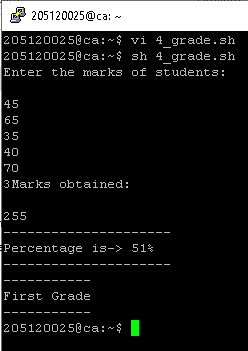
echo "----"

echo "Fail"

echo "----"

fi

Output:-



5. Write a shell script to read two integer numbers and perform basic arithmetic operations based on user’s choice (use ‘case’ structure).

Source Code:-

# !/bin/bash

# Take user Input

echo "Enter Two numbers : "

read a

read b

# Input type of operation

echo "Enter Choice :"

echo "1. Addition"

echo "2. Subtraction"

echo "3. Multiplication"

echo "4. Division"

echo "Enter Your Choice"

read ch

# Switch Case to perform

# calulator operations

case $ch in

1)res=`echo $a + $b | bc`

;;

2)res=`echo $a - $b | bc`

;;

3)res=`echo $a \\* $b | bc`

;;

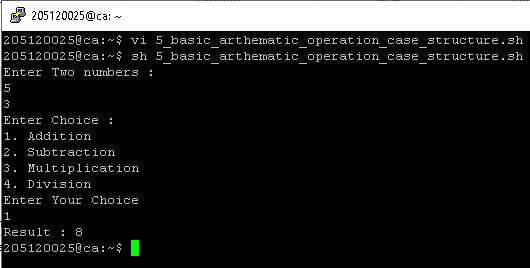
4)res=`echo "scale=2; $a / $b" | bc`

;;

esac

echo "Result : $res"

Output:-



6. Write a shell script to find the sum of first ‘N’ Natural Numbers (use ‘while’ structure).

Source Code:-

echo "Enter a number:"

read num

sum=0

while [ $num -ne 0 ]

do

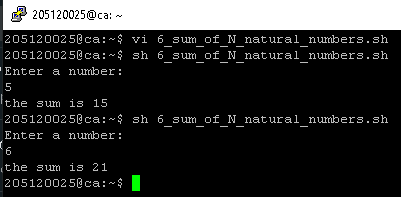
sum=`expr $sum + $num`

num=`expr $num - 1`

done

echo "the sum is $sum "

Output:-



7. Write a shell script to find the sum of first ‘N’ numbers in Fibonacci series (use ‘for’ structure).

Source Code:-

echo "Enter value of n: "

read n

a=-1

b=1

c=$(($a+$b))

sum=0

i=1

while [ $i -le $n ]

do

sum=$((sum+c))

echo $c

a=$((b))

b=$((c))

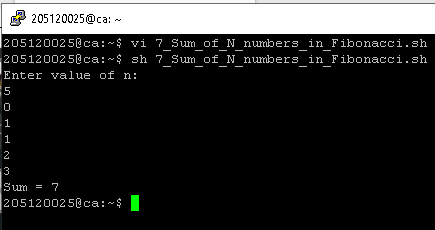
c=$((a+b))

i=$((i+1))

done

echo "Sum = $sum"

Output:-



8. Write a shell script to print a given number in reverse order and sum of the individual digits.

Source Code:-

echo -n "Enter number : "

read n

# store single digit

sd=0

# store number in reverse order

rev=""

sum=""

# store original number

on=$n

# use while loop to caclulate the sum of all digits

while [ $n -gt 0 ]

do

sd=$(( $n % 10 )) # get Remainder

n=$(( $n / 10 )) # get next digit

sum=$(($sum + $sd))

# store previoues number and current digit in rev

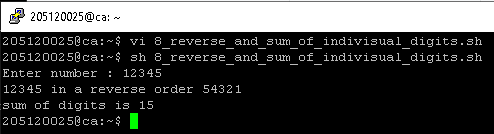
rev=$( echo ${rev}${sd} )

done

echo "$on in a reverse order $rev"

echo "sum of digits is $sum"

Output:-



9. Write a shell script to read two strings and display whether it is equal, not equal, null strings or string with special characters.

Source Code:-

#!bin/sh

read -p " enter string 1 : " str1

read -p " enter string 2 : " str2

if [ $str1 = $str2 ]

then echo " strings are equal !"

else

echo " strings are not equal "

fi

if [ -z "$str1" ]

then echo "string 1 is null "

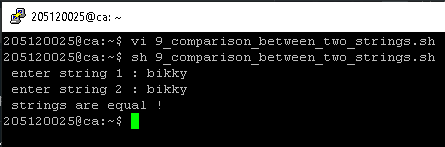
fi

if [ -z "$str2" ]

then echo "string 2 is null "

fi

Output:-



10. Write a shell script to accept one integer argument and print its multiplication table.

Source Code:-

#!/bin/bash

echo "Enter Number "

read n

i=1

while [ $i -le 10 ]

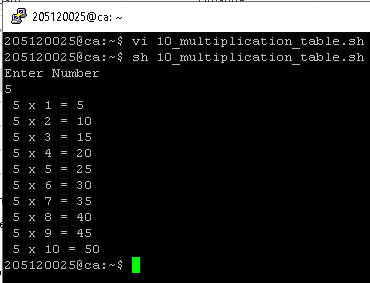
do

echo " $n x $i = `expr $n \\* $i`"

i=`expr $i + 1`

done

Output:-



11. Write a shell script, which accepts any number of arguments and prints them in the Reverse order. (For example, if the script is passed A B C as arguments, then execution should produce C B A on the standard output).

Source Code:-

a=$#

echo "Number of arguments are" $a

x=$\*

c=$a

res=''

while [ 1 -le $c ]

do

c=`expr $c - 1`

shift $c

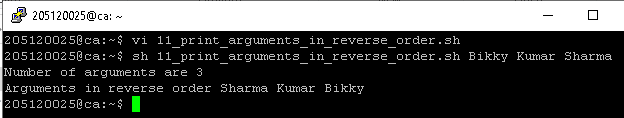
res=$res' '$1

set $x

done

echo Arguments in reverse order $res

Output:-

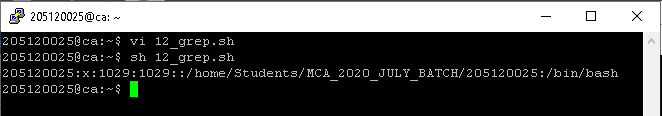


12. Write a Shell Script that makes use of grep to isolate the line in /etc/passwd that contains your login details.

Source Code:-

grep "^`whoami`:" /etc/passwd

Output:-



13. Write a shell script to display all files in the /home/YourLoginName subdirectory as well as display the type of all files.

Source Code:-

for file in \*

do

if [ -d $file ]

then echo "$file is a dir "

elif [ -f $file ]

then echo "$file is a file "

elif [ -p $file ]

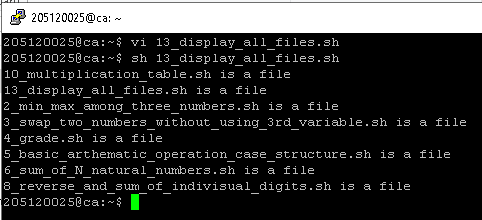
then echo "$file is a pipe "

else echo "$file is a hyperlink "

fi

done

Output:-



14. Using shell script, display the contents of the present working directory. If it is an ordinary file print its permission and change the permissions to r--r--r—

Source Code:-

# !/bin/bash

for item in \*

do

if [ -f $item ]

then

echo "----------------$item----------------"

if [ -x $item ]

then

echo "File in Executable mode"

chmod -x $item

echo "Executable permission Removed!"

fi

if [ -w $item ]

then

echo "File in Write mode"

chmod -w $item

echo "Write permission Removed!"

fi

if [ -r $item ]

then

echo "Already in read mode(r--r--r--)"

else

chmod +r $item

echo "Now the read permission granted "

fi

echo "final permission"

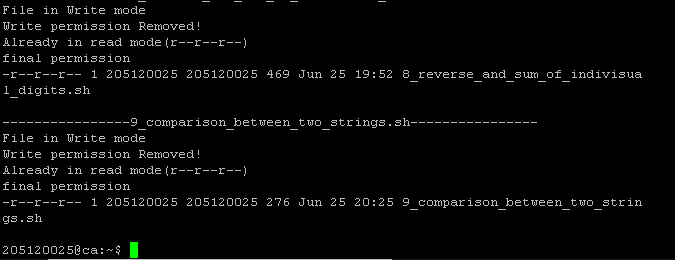
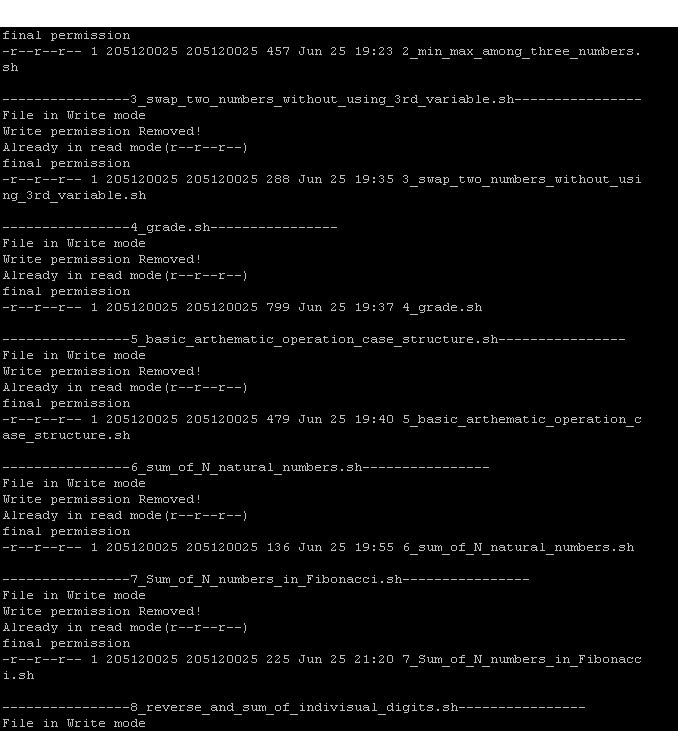
ls -al $item

fi

echo

done

Output:-

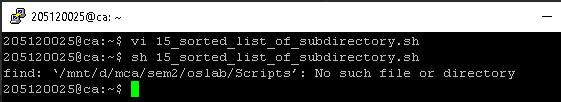


15. Use find, grep and sort to display a sorted list of all files in the /home/YourLoginName subdirectory that contains the word “hello” somewhere inside them.

Source Code:-

find /mnt/d/mca/sem2/oslab/Scripts -type f -print0 | xargs -0 grep -li "hello" | sort

Output:-

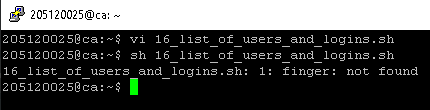


16. Write a shell script to produce a list of users and their login shells.

Source Code:-

finger $USER | grep 'Shell:\*' | cut -f3 -d ":"

Output:-



17. Write a C program to kill a process by specifying its name rather than its PID.

Source Code:-

#include<stdio.h>

#include<string.h>

main()

{

char cmd[50],cmd1[50],cmd2[50],log[50],pname[50],pid[50];

FILE \* fp;

system("rm newpro");

system("rm data");

printf("enter ur login name\n");

fgets(log,sizeof(log),stdin);

strcpy(cmd,"ps -aux | grep ");

strcat(cmd,log);

system(cmd);

printf("enter the name of the process u want to terminate\n");

scanf("%s",pname);

strcpy(cmd1,"ps -a | grep ");

strcat(cmd1,pname);

strcat(cmd1," > newpro");

system(cmd1);

system("cut -f2 -d' ' newpro > data");

fp=fopen("data","r");

fscanf(fp,"%s",pid);

strcpy(cmd2,"kill ");

strcat(cmd2,pid);

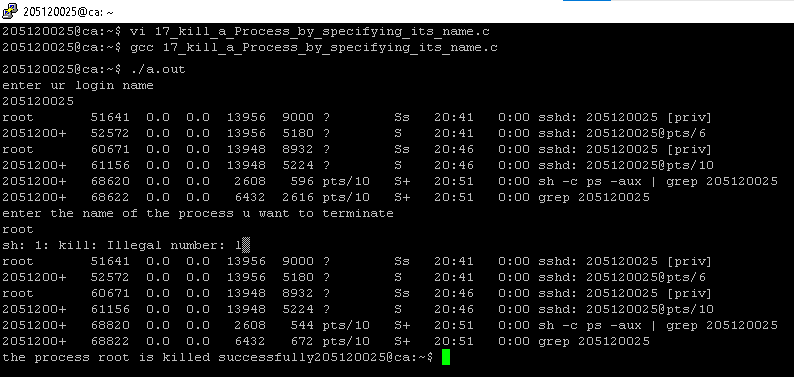
system(cmd2);

system(cmd);

printf("the process %s is killed successfully",pname);

}

Output:-



18. Create a file with few lines, Write a C program to read the file and delete the spaces more than one in the file (use UNIX file API’s).

Source Code:-

#include<stdio.h>

#include<ctype.h>

int main()

{

FILE \* pfile;

int a;

printf("\n Remove the spaces between two words :\n");

printf("-----------------------------------------\n");

// file.txt contain : the quick brown fox jumps over the lazy dog

pfile=fopen ("file.txt","r");

printf(" The content of the file is :\n The quick brown fox jumps over the lazy dog\n\n");

printf(" After removing the spaces the content is : \n");

if (pfile)

{

do {

a = fgetc (pfile);

if ( isgraph(a) ) putchar (a);

} while (a != EOF);

fclose (pfile);

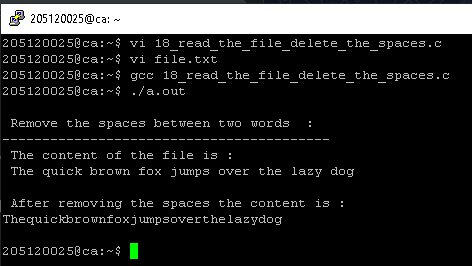
}

printf("\n\n");

return 0;

}

Output:-



19. Implement a C program to list the users who have logged in more than once.

Source Code:-

// Implement a C program to list the users who have logged in more than once.

#include <stdio.h>

#include <sys/utsname.h>

#include <utmp.h>

int main(void)

{

struct utmp \*n;

setutent();

n = getutent();

while (n)

{

if (n->ut\_type == USER\_PROCESS)

{

printf("%9s%12s (%s)\n", n->ut\_user, n->ut\_line, n->ut\_host);

}

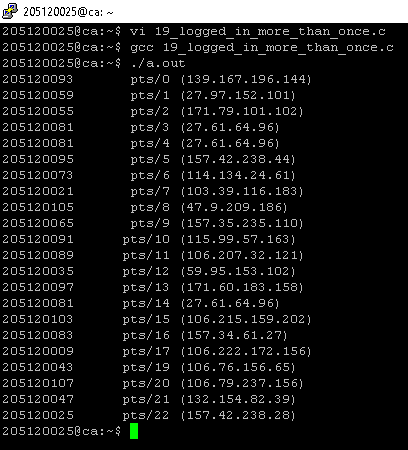
n = getutent();

}

return 0;

}

Output:-



20. Write a C program which renames all .txt files as .text files.

Source Code:-

// 20. Write a C program which renames all .txt files as .text files.

#include <stdio.h>

#include <string.h>

int main()

{

FILE \*fp;

int i;

char temp[50], cmd[20];

system("dir");

system("dir > listforques20");

fp = fopen("listforques20", "r");

while (!feof(fp))

{

fscanf(fp, "%s", temp);

i = 0;

strcpy(cmd, "mv ");

while (temp[i] != '.')

{

i++;

}

if (temp[i + 1] == 't' && temp[i + 2] == 'x' && temp[i + 3] == 't')

{

strcat(cmd, temp);

strcat(cmd, " ");

temp[i + 2] = 'e';

temp[i + 3] = 'x';

temp[i + 4] = 't';

temp[i + 5] = '\0';

strcat(cmd, temp);

//printf("%s\n",cmd);

system(cmd);

}

//printf(" %s \n",temp);

}

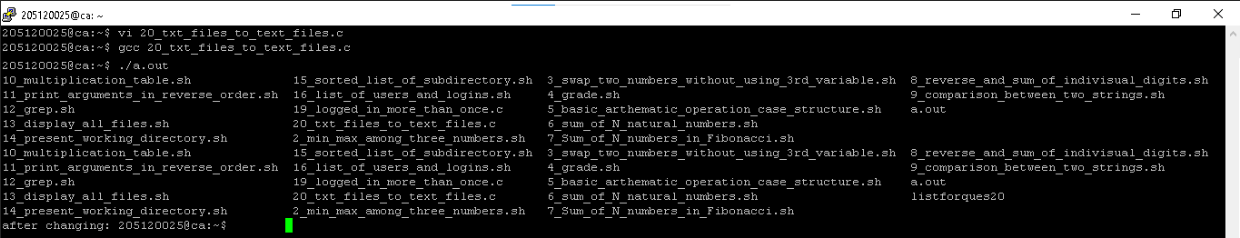
printf("after changing: ");

system("dir");

return 0;

}

Output:-



21. Implement a C program that reports the number of file names in the current working directory that consist of exactly five characters.

Source Code:-

#include <stdio.h>

#include <string.h>

#include<stdlib.h>

int main()

{

FILE \*fp;

int i;

char temp[50], cmd[20];

system("dir > listforques21");

fp = fopen("listforques21", "r");

while (!feof(fp))

{

fscanf(fp, "%s", temp);

if (strlen(temp) == 5)

{

printf(" %s \n", temp);

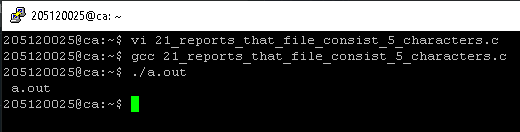
}

}

return 0;

}

Output:-



22. Write Programs to a) Report the behaviour of the OS to get the CPU type and model, kernel version. b) Get the amount of memory configured into the computer, amount of memory currently available.

Source Code A:-

(A)>>>>>>>>

// 22. Write Programs to

// a) Report the behavior of the OS to get the CPU type and model, kernel version.

#include<stdio.h>

#include<stdlib.h>

int main()

{

system("cat /proc/cpuinfo | grep model\\ name");

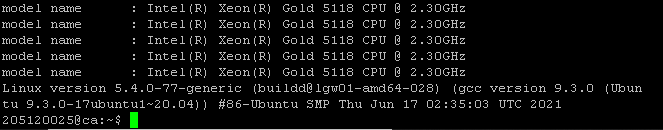
system("cat /proc/version");

return 0;

}

Output A:-





Source Code B:-

(B)>>>>>>

// 22. Write Programs to

// b) Get the amount of memory configured into the computer, amount of memory currently available.

#include <stdio.h>

#include <stdlib.h>

int main()

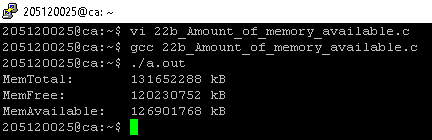
{

system("cat /proc/meminfo | grep Mem");

return 0;

}

Output:-



23. Write a program to a create child process and display the process ID of parent and child processes.

Source Code:-

// Write a program to a create child process and display the process ID of parent and child processes.

#include <stdio.h>

#include <unistd.h>

#include <sys/types.h>

int main(void)

{

pid\_t childpid;

childpid = fork();

if (childpid == -1)

{

perror("Failed to fork");

return 1;

}

if (childpid == 0)

/\* child code \*/

printf("I am child %ld\n", (long)getpid());

else

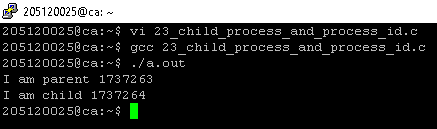
/\* parent code \*/

printf("I am parent %ld\n", (long)getpid());

return 0;

}

Output:-



24. Write a program to demonstrate the implementation of Inter Process Communication (IPC) "who | grep YourLoginName" using pipes.

Source Code:-

// 24. Write a program to demonstrate the implementation of Inter Process Communication (IPC) "who | grep YourLoginName" using pipes.

#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

int main()

{

int pid;

int fd[2];

pipe(fd);

pid = fork();

if (pid == -1)

{

perror("fork");

exit(-1);

}

if (pid)

{

close(0);

dup(fd[0]);

close(fd[1]);

execl("/usr/bin/wc", "wc", "-l", (char \*)0);

close(fd[0]);

}

else

{

close(1);

dup(fd[1]);

close(fd[0]);

execl("/usr/bin/who", "who", (char \*)0);

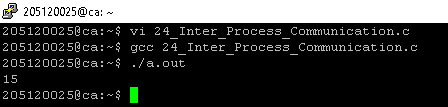
close(fd[1]);

}

return 0;

}

Output:-



25. Write a program to demonstrate the implementation of Inter Process Communication (IPC) using Message Queues.

Source Code:-

// 25. Write a program to demonstrate the implementation of Inter Process Communication (IPC) using Message Queues.

#include <stdio.h>

#include <sys/ipc.h>

#include <sys/msg.h>

// structure for message queue

struct msg\_buffer

{

long msg\_type;

char msg[100];

} message;

int main()

{

key\_t my\_key;

int msg\_id;

my\_key = ftok("progfile", 65); //create unique key

msg\_id = msgget(my\_key, 0666 | IPC\_CREAT); //create message queue and return id

message.msg\_type = 1;

printf("Write Message : ");

fgets(message.msg, 100, stdin);

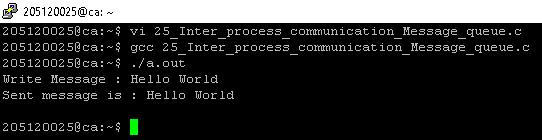
msgsnd(msg\_id, &message, sizeof(message), 0); //send message

printf("Sent message is : %s \n", message.msg);

return 0;

}

Output:-



26. Write a program to demonstrate the implementation of Inter Process Communication (IPC) using shared memory.

Source Code:-

// 26. Write a program to demonstrate the implementation of Inter Process Communication (IPC) using shared memory.

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <sys/types.h>

#define SEGSIZE 100

int main(int argc, char \*argv[])

{

int shmid, cntr;

key\_t key;

char \*segptr;

char buff[] = "poooda......";

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

if ((shmid = shmget(key, SEGSIZE, IPC\_CREAT | IPC\_EXCL | 0666)) == -1)

{

if ((shmid = shmget(key, SEGSIZE, 0)) == -1)

{

perror("shmget");

exit(1);

}

}

else

{

printf("Creating a new shared memory seg \n");

printf("SHMID:%d", shmid);

}

system("ipcs –m");

if ((segptr = (char \*)shmat(shmid, 0, 0)) == (char \*)-1)

{

perror("shmat");

exit(1);

}

printf("Writing data to shared memory...\n");

strcpy(segptr, buff);

printf("DONE\n");

printf("Reading data from shared memory...\n");

printf("DATA:-%s\n", segptr);

printf("DONE\n");

printf("Removing shared memory Segment...\n");

if (shmctl(shmid, IPC\_RMID, 0) == -1)

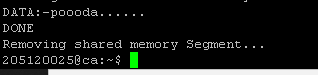
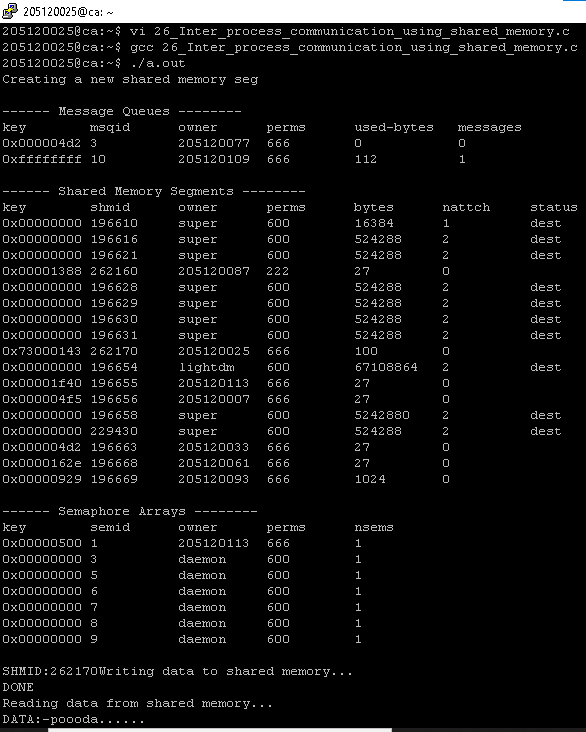
printf("Can‟t Remove Shared memory Segment...\n");

else

printf("Removed Successfully");

}

Output:-



27. Write a program to create a thread and let the thread check whether the given number is prime or not.

Source Code:-

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <pthread.h>

void thread\_check\_if\_number\_is\_prime(void ) {

int count = 0, i, number;

printf("Enter the number to be checked: ");

scanf("%d", &number);

for(i = 2; i < number; i++)

if(number % i == 0)

++count;

if(count == 0)

printf("The number is a prime number!\n");

else

printf("The number is not a prime number!\n");

return NULL;

}

int main() { pthread\_t thread\_id;

printf("Thread is starting now!\n");

pthread\_create(&thread\_id, NULL, thread\_check\_if\_number\_is\_prime, NULL);

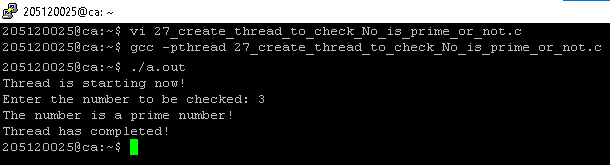
pthread\_join(thread\_id, NULL);

printf("Thread has completed!\n");

exit(0);

}

Output:-



28. Implement FCFS, SJF, Priority and Round– Robin process scheduling algorithms.

Source Code FCFS:-

FCFS>>>>>

#include <stdio.h>

int main()

{

int bt[20], wt[20], tat[20], i, n;

float wtavg, tatavg;

printf("\nEnter the number of processes:");

scanf("%d", &n);

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

{

printf("\nEnter Burst Time for Process %d:", i);

scanf("%d", &bt[i]);

}

wt[0] = wtavg = 0;

tat[0] = tatavg = bt[0];

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

{

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

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

wtavg = wtavg + wt[i];

tatavg = tatavg + tat[i];

}

printf("\t PROCESS \tBURST TIME \t WAITING TIME\t TURNAROUND TIME\n");

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

printf("\n\t P%d \t\t %d \t\t %d \t\t%d", i, bt[i], wt[i], tat[i]);

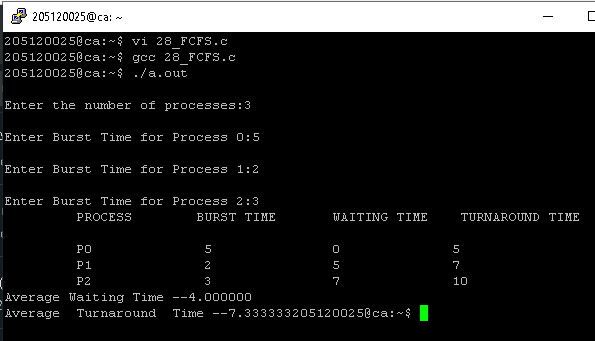
printf("\nAverage Waiting Time --%f", wtavg / n);

printf("\nAverage Turnaround Time --%f", tatavg / n);

return 0;

}

Output FCFS:-



Source Code SJF:-

SJF>>>>

#include <stdio.h>

int main()

{

int p[20], bt[20], wt[20], tat[20], i, k, n, temp;

float wtavg, tatavg;

printf("\nEnter the number of processes --");

scanf("%d", &n);

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

{

p[i] = i;

printf("Enter Burst Time for Process %d --", i);

scanf("%d", &bt[i]);

}

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

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

if (bt[i] > bt[k])

{

temp = bt[i];

bt[i] = bt[k];

bt[k] = temp;

temp = p[i];

p[i] = p[k];

p[k] = temp;

}

wt[0] = wtavg = 0;

tat[0] = tatavg = bt[0];

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

{

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

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

wtavg = wtavg + wt[i];

tatavg = tatavg + tat[i];

}

printf("\n\t PROCESS \tBURST TIME \t WAITING TIME\t TURNAROUND TIME\n");

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

printf("\n\t P%d \t\t %d \t\t %d \t\t %d", p[i], bt[i], wt[i], tat[i]);

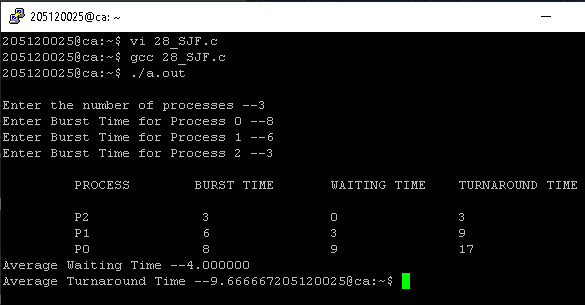
printf("\nAverage Waiting Time --%f", wtavg / n);

printf("\nAverage Turnaround Time --%f", tatavg / n);

return 0;

}

Output SJF:-



Source Code Priority:-

Priority>>>>

#include <stdio.h>

int main()

{

int p[20], bt[20], pri[20], wt[20], tat[20], i, k, n, temp;

float wtavg, tatavg;

printf("Enter the number of processes ---");

scanf("%d", &n);

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

{

p[i] = i;

printf("Enter the Burst Time & Priority of Process %d ---", i);

scanf("%d%d", &bt[i], &pri[i]);

}

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

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

if (pri[i] > pri[k])

{

temp = p[i];

p[i] = p[k];

p[k] = temp;

temp = bt[i];

bt[i] = bt[k];

bt[k] = temp;

temp = pri[i];

pri[i] = pri[k];

pri[k] = temp;

}

wtavg = wt[0] = 0;

tatavg = tat[0] = bt[0];

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

{

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

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

wtavg = wtavg + wt[i];

tatavg = tatavg + tat[i];

}

printf("\nPROCESS\t\tPRIORITY\tBURST TIME\tWAITING TIME\tTURNAROUND TIME");

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

printf("\n%d \t\t %d \t\t %d \t\t %d \t\t %d ", p[i], pri[i], bt[i], wt[i], tat[i]);

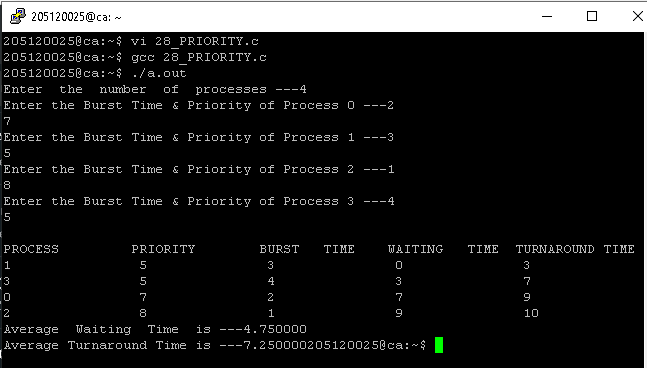
printf("\nAverage Waiting Time is ---%f", wtavg / n);

printf("\nAverage Turnaround Time is ---%f", tatavg / n);

return 0;

}

Output Priority:-



Source Code Round Robin:-

Round Robin>>>>

// 28. Implement FCFS, SJF, Priority and Round– Robin process scheduling algorithms.

// Round Robin

#include <stdio.h>

int main()

{

int i, j, n, bu[10], wa[10], tat[10], t, ct[10], max;

float awt = 0, att = 0, temp = 0;

printf("Enter the no of processes --");

scanf("%d", &n);

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

{

printf("\nEnter Burst Time for process %d --", i + 1);

scanf("%d", &bu[i]);

ct[i] = bu[i];

}

printf("\nEnter the size of time slice --");

scanf("%d", &t);

max = bu[0];

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

if (max < bu[i])

max = bu[i];

for (j = 0; j < (max / t) + 1; j++)

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

if (bu[i] != 0)

if (bu[i] <= t)

{

tat[i] = temp + bu[i];

temp = temp + bu[i];

bu[i] = 0;

}

else

{

bu[i] = bu[i] - t;

temp = temp + t;

}

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

{

wa[i] = tat[i] - ct[i];

att += tat[i];

awt += wa[i];

}

printf("\nThe Average Turnaround time is --%f", att / n);

printf("\nThe Average Waiting time is --%f ", awt / n);

printf("\n\tPROCESS\t BURST TIME \t WAITING TIME\tTURNAROUND TIME\n");

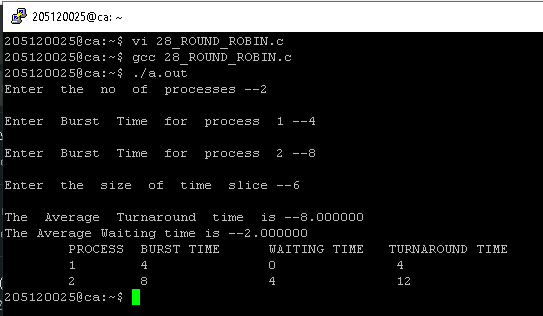
for (i = 0; i < n; i++)

printf("\t%d \t %d \t\t %d \t\t %d \n", i + 1, ct[i], wa[i], tat[i]);

return 0;

}

Output Round Robin:-



29. Write a program to perform a tidy exit on receipt of an interrupt signal.

Source Code:-

#include <stdio.h>

#include <stdlib.h>

#include <signal.h>

FILE \*temp\_file;

void leave(int sig);

int main()

{

(void)signal(SIGINT, leave);

temp\_file = fopen("tmp", "w");

for (;;)

{

/\*

\* Do things....

\*/

printf("Ready...\n");

(void)getchar();

}

/\* can't get here ... \*/

exit(EXIT\_SUCCESS);

return 0;

}

// SIGHUP 1 Hang up detected on controlling terminal or death of controlling process

// SIGINT 2 Issued if the user sends an interrupt signal (Ctrl + C)

// SIGQUIT 3 Issued if the user sends a quit signal (Ctrl + D)

// SIGFPE 8 Issued if an illegal mathematical operation is attempted

// SIGKILL 9 If a process gets this signal it must quit immediately and will not perform any clean-up operations

// SIGALRM 14 Alarm clock signal (used for timers)

// SIGTERM 15 Software termination signal (sent by kill by default)

// kill -l

void leave(int sig)

{

printf("\nSIGINT Recieved, Exiting");

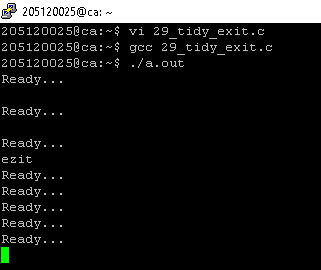
fprintf(temp\_file, "\nInterrupted..\n");

fclose(temp\_file);

exit(sig);

}

Output:-



30. Implement a) Binary Semaphore b) Counting Semaphore.

Source Code Binary Semaphore:

#include <stdio.h>

#include <pthread.h>

#include <semaphore.h>

#include <unistd.h>

sem\_t mutex;

void\* thread(void\* arg)

{

//wait

sem\_wait(&mutex);

printf("\nEntered..\n");

//critical section

sleep(4);

//signal

printf("\nJust Exiting...\n");

sem\_post(&mutex);

}

int main()

{

sem\_init(&mutex, 0, 1);

pthread\_t t1,t2;

pthread\_create(&t1,NULL,thread,NULL);

sleep(2);

pthread\_create(&t2,NULL,thread,NULL);

pthread\_join(t1,NULL);

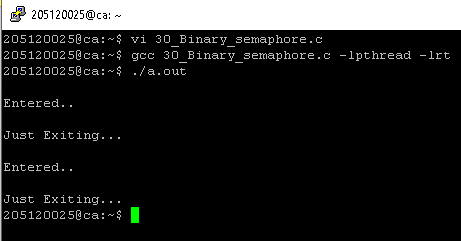
pthread\_join(t2,NULL);

sem\_destroy(&mutex);

return 0;

}

Output:-



31. Write a program to demonstrate the implementation of Producer and Consumer problem.

Source Code:-

// 31. Write a program to demonstrate the implementation of Producer and Consumer problem.

#include <stdio.h>

int main()

{

int buffer[10], bufsize, in, out, produce, consume, choice = 0;

in = 0;

out = 0;

bufsize = 10;

while (choice != 3)

{

printf("\n 1.Produce\t 2.Consume\t 3.Exit");

printf("\nEnter your choice: ");

scanf("%d", &choice);

switch (choice)

{

case 1:

if ((in + 1) % bufsize == out)

printf("\nBuffer is Full");

else

{

printf("\nEnter the value: ");

scanf("%d", &produce);

buffer[in] = produce;

in = (in + 1) % bufsize;

}

break;

case 2:

if (in == out)

printf("\nBuffer is Empty\n");

else

{

consume = buffer[out];

printf("\nThe consumed value is %d\n", consume);

out = (out + 1) % bufsize;

}

break;

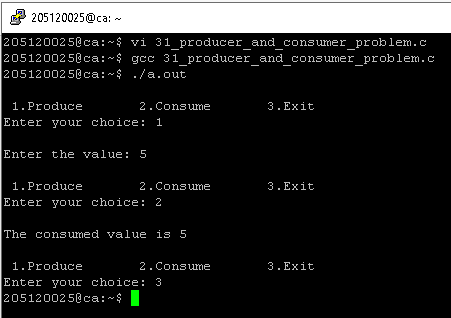
}

}

return 0;

}

Output:-



32. Write a program to implement Reader – Writer’s problem.

Source Code:-

// 32. Write a program to implement Reader – Writer’s problem.

#include <pthread.h>

#include <semaphore.h>

#include <stdio.h>

sem\_t wrt;

pthread\_mutex\_t mutex;

int cnt = 1;

int numreader = 0;

void \*writer(void \*wno)

{

sem\_wait(&wrt);

cnt = cnt \* 2;

printf("Writer %d modified cnt to %d\n", (\*((int \*)wno)), cnt);

sem\_post(&wrt);

}

void \*reader(void \*rno)

{

// Reader acquire the lock before modifying numreader

pthread\_mutex\_lock(&mutex);

numreader++;

if (numreader == 1)

{

sem\_wait(&wrt); // If this id the first reader, then it will block the writer

}

pthread\_mutex\_unlock(&mutex);

// Reading Section

printf("Reader %d: read cnt as %d\n", \*((int \*)rno), cnt);

// Reader acquire the lock before modifying numreader

pthread\_mutex\_lock(&mutex);

numreader--;

if (numreader == 0)

{

sem\_post(&wrt); // If this is the last reader, it will wake up the writer.

}

pthread\_mutex\_unlock(&mutex);

}

int main()

{

pthread\_t read[10], write[5];

pthread\_mutex\_init(&mutex, NULL);

sem\_init(&wrt, 0, 1);

int a[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}; //Just used for numbering the producer and consumer

for (int i = 0; i < 10; i++)

{

pthread\_create(&read[i], NULL, (void \*)reader, (void \*)&a[i]);

}

for (int i = 0; i < 5; i++)

{

pthread\_create(&write[i], NULL, (void \*)writer, (void \*)&a[i]);

}

for (int i = 0; i < 10; i++)

{

pthread\_join(read[i], NULL);

}

for (int i = 0; i < 5; i++)

{

pthread\_join(write[i], NULL);

}

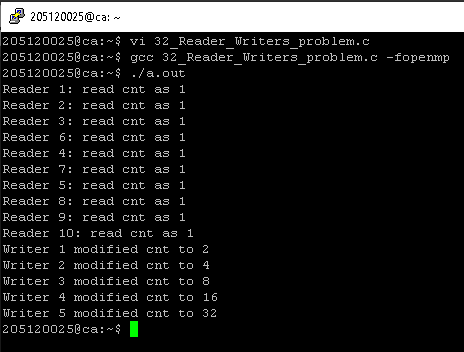
pthread\_mutex\_destroy(&mutex);

sem\_destroy(&wrt);

return 0;

}

Output:-



33. Write a program to implement Dining Philosopher’s problem. Implement Banker’s algorithm.

Source Code:-

//Write a program to implement Dining Philosopher’s problem. Implement Banker’s algorithm.

#include <pthread.h>

#include <semaphore.h>

#include <stdio.h>

#include<unistd.h>

#define N 5

#define THINKING 2

#define HUNGRY 1

#define EATING 0

#define LEFT (phnum + 4) % N

#define RIGHT (phnum + 1) % N

int state[N];

int phil[N] = { 0, 1, 2, 3, 4 };

sem\_t mutex;

sem\_t S[N];

void test(int phnum)

{

if (state[phnum] == HUNGRY

&& state[LEFT] != EATING

&& state[RIGHT] != EATING) {

// state that eating

state[phnum] = EATING;

sleep(2);

printf("Philosopher %d takes fork %d and %d\n",

phnum + 1, LEFT + 1, phnum + 1);

printf("Philosopher %d is Eating\n", phnum + 1);

// sem\_post(&S[phnum]) has no effect

// during takefork

// used to wake up hungry philosophers

// during putfork

sem\_post(&S[phnum]);

}

}

// take up chopsticks

void take\_fork(int phnum)

{

sem\_wait(&mutex);

// state that hungry

state[phnum] = HUNGRY;

printf("Philosopher %d is Hungry\n", phnum + 1);

// eat if neighbours are not eating

test(phnum);

sem\_post(&mutex);

// if unable to eat wait to be signalled

sem\_wait(&S[phnum]);

sleep(1);

}

// put down chopsticks

void put\_fork(int phnum)

{

sem\_wait(&mutex);

// state that thinking

state[phnum] = THINKING;

printf("Philosopher %d putting fork %d and %d down\n",

phnum + 1, LEFT + 1, phnum + 1);

printf("Philosopher %d is thinking\n", phnum + 1);

test(LEFT);

test(RIGHT);

sem\_post(&mutex);

}

void\* philospher(void\* num)

{

while (1) {

int\* i = num;

sleep(1);

take\_fork(\*i);

sleep(0);

put\_fork(\*i);

}

}

int main()

{

int i;

pthread\_t thread\_id[N];

// initialize the semaphores

sem\_init(&mutex, 0, 1);

for (i = 0; i < N; i++)

sem\_init(&S[i], 0, 0);

for (i = 0; i < N; i++) {

// create philosopher processes

pthread\_create(&thread\_id[i], NULL,

philospher, &phil[i]);

printf("Philosopher %d is thinking\n", i + 1);

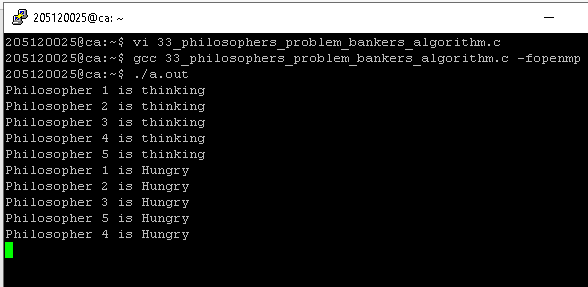
}

for (i = 0; i < N; i++)

pthread\_join(thread\_id[i], NULL);

}

Output:-



34. Implement the First Fit, Best Fit and Worst Fit file allocation strategy.

Source Code First Fit:-

#include <stdio.h>

void main()

{

//declaration and initialization of variables

int blocksize[10], process\_size[10], block\_no, process\_no, flags[10], allocate[10], i, j;

for (i = 0; i < 10; i++)

{

flags[i] = 0;

allocate[i] = -1;

}

printf("\nEnter the number of blocks: ");

scanf("%d", &block\_no);

//code to define block size

printf("\nEnter block size: ");

for (i = 0; i < block\_no; i++)

scanf("%d", &blocksize[i]);

//code to define processes

printf("\nEnter the number of processes: ");

scanf("%d", &process\_no);

printf("\nEnter size of each process: ");

for (i = 0; i < process\_no; i++)

{

scanf("%d", &process\_size[i]);

}

//allocate as per first fit

for (i = 0; i < process\_no; i++)

for (j = 0; j < block\_no; j++)

if (flags[j] == 0 && blocksize[j] >= process\_size[i])

{

allocate[j] = i;

flags[j] = 1;

break;

}

//code to show details for allocation

printf("\nBlock no.\tsize\t\tprocess no.\t\tsize");

for (i = 0; i < block\_no; i++)

{

printf("\n%d\t\t%d\t\t", i + 1, blocksize[i]);

if (flags[i] == 1)

printf("%d\t\t\t%d\n", allocate[i] + 1, process\_size[allocate[i]]);

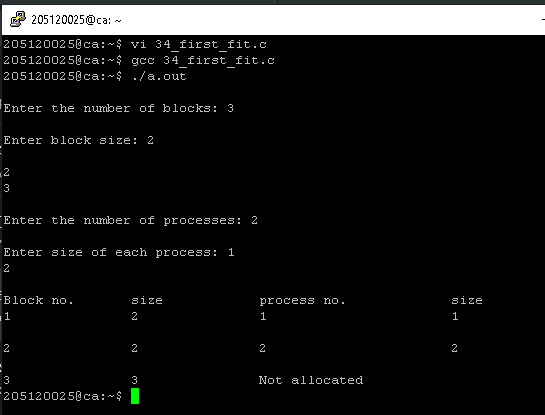
else

printf("Not allocated\n");

}

}

Output First Fit:-



Source Code Best Fit:-

#include <stdio.h>

// main function

void main()

{

// declaration and initialization of variables

int fragment[20], b[20], p[20], i, j, n\_b, n\_p, tem, low = 9999;

static int barray[20], parray[20];

printf("Memory Management Scheme - Best Fit\n");

//input number of processes

printf("Enter the number of processes:\t");

scanf("%d", &n\_p);

//input number of blocks

printf("\nEnter the number of blocks:");

scanf("%d", &n\_b);

//input number of blocks

printf("\nEnter the size of the blocks:-\n");

for (i = 1; i <= n\_b; i++)

{

printf("Block no.%d:", i);

scanf("%d", &b[i]);

}

//input the size of process

printf("\nEnter the size of the processes :-\n");

for (i = 1; i <= n\_p; i++)

{

printf("Process no.%d:", i);

scanf("%d", &p[i]);

}

for (i = 1; i <= n\_p; i++)

{

for (j = 1; j <= n\_b; j++)

{

if (barray[j] != 1)

{

tem = b[j] - p[i];

if (tem >= 0)

if (low > tem)

{

parray[i] = j;

low = tem;

}

}

}

fragment[i] = low;

barray[parray[i]] = 1;

low = 10000;

}

//Print the result

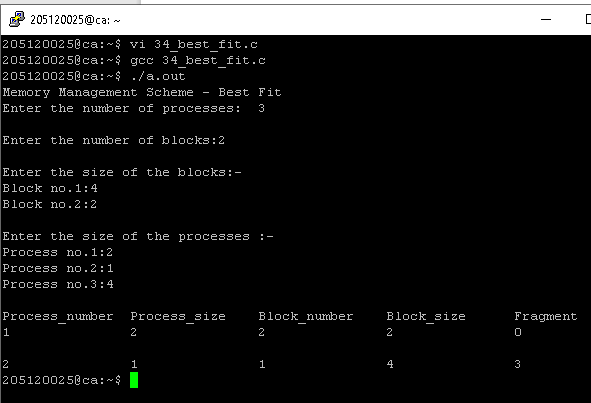
printf("\nProcess\_number \tProcess\_size\tBlock\_number \tBlock\_size\tFragment");

for (i = 1; i <= n\_p && parray[i] != 0; i++)

printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d\n", i, p[i], parray[i], b[parray[i]], fragment[i]);

}

Output Best Fit:-



Source Code Worst Fit:-

//34. Program code for Worst Fit Memory management Scheme using C:

#include <stdio.h>

int main()

{

int frag[10], blocks[10], process[10];

int m, n, block\_number, process\_number, temp, top = 0;

static int block\_arr[10], process\_arr[10];

//Fill the number of blocks

printf("\nEnter the Total Number of Blocks:\t");

scanf("%d", &block\_number);

//Fill the number of processes

printf("Enter the Total Number of Processes:\t");

scanf("%d", &process\_number);

//Fill the size of the blocks

printf("\nEnter the Size of the Blocks:\n");

for (m = 0; m < block\_number; m++)

{

printf("Block No.[%d]:\t", m);

scanf("%d", &blocks[m]);

}

//Fill the size of the processes

printf("Enter the Size of the processes:\n");

for (m = 0; m < process\_number; m++)

{

printf("process No.[%d]:\t", m);

scanf("%d", &process[m]);

}

//memory allocation to the processes

for (m = 0; m < process\_number; m++)

{

for (n = 0; n < process\_number; n++)

{

if (block\_arr[n] != 1)

{

temp = blocks[n] - process[m];

if (temp >= 0)

{

if (top < temp)

{

process\_arr[m] = n;

top = temp;

}

}

}

frag[m] = top;

block\_arr[process\_arr[m]] = 1;

top = 0;

}

}

//printing result after memory allocation

printf("\nProcess Number\tProcess Size\tBlock Number\tBlock Size\tFragment");

for (m = 0; m < process\_number; m++)

{

printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d", m, process[m], process\_arr[m], blocks[process\_arr[m]], frag[m]);

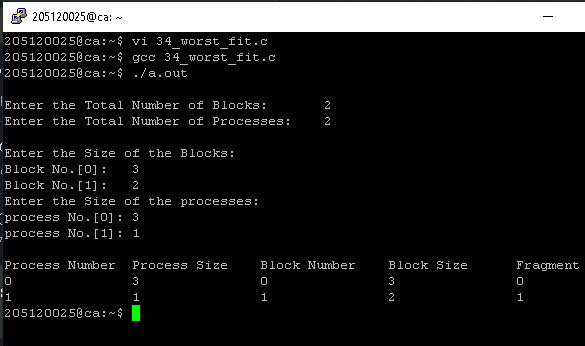
}

printf("\n");

return 0;

}

Output Worst Fit:-



35. Implement FIFO, Optimal, LRU and LFU page replacement algorithms.

Source Code:-

#include <stdio.h>

int n, nf;

int in[100];

int p[50];

int hit = 0;

int i, j, k;

int pgfaultcnt = 0;

void getData()

{

printf("\nEnter length of page reference sequence:");

scanf("%d", &n);

printf("\nEnter the page reference sequence:");

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

{

scanf("%d", &in[i]);

}

printf("\nEnter no of frames:");

scanf("%d", &nf);

}

void initialize()

{

pgfaultcnt = 0;

for (i = 0; i < nf; i++)

p[i] = 9999;

}

int isHit(int data)

{

hit = 0;

for (j = 0; j < nf; j++)

{

if (p[j] == data)

{

hit = 1;

break;

}

}

return hit;

}

int getHitIndex(int data)

{

int hitind;

for (k = 0; k < nf; k++)

{

if (p[k] == data)

{

hitind = k;

break;

}

}

return hitind;

}

void dispPages()

{

for (k = 0; k < nf; k++)

{

if (p[k] != 9999)

printf(" %d", p[k]);

}

}

void dispPgFaultCnt()

{

printf("\nTotal no of page faults:%d", pgfaultcnt);

}

void fifo()

{

initialize();

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

{

printf("\nFor %d :", in[i]);

if (isHit(in[i]) == 0)

{

for (k = 0; k < nf - 1; k++)

p[k] = p[k + 1];

p[k] = in[i];

pgfaultcnt++;

dispPages();

}

else

printf("No page fault");

}

dispPgFaultCnt();

}

void optimal()

{

initialize();

int near[50];

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

{

printf("\nFor %d :", in[i]);

if (isHit(in[i]) == 0)

{

for (j = 0; j < nf; j++)

{

int pg = p[j];

int found = 0;

for (k = i; k < n; k++)

{

if (pg == in[k])

{

near[j] = k;

found = 1;

break;

}

else

found = 0;

}

if (!found)

near[j] = 9999;

}

int max = -9999;

int repindex;

for (j = 0; j < nf; j++)

{

if (near[j] > max)

{

max = near[j];

repindex = j;

}

}

p[repindex] = in[i];

pgfaultcnt++;

dispPages();

}

else

printf("No page fault");

}

dispPgFaultCnt();

}

void lru()

{

initialize();

int least[50];

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

{

printf("\nFor %d :", in[i]);

if (isHit(in[i]) == 0)

{

for (j = 0; j < nf; j++)

{

int pg = p[j];

int found = 0;

for (k = i - 1; k >= 0; k--)

{

if (pg == in[k])

{

least[j] = k;

found = 1;

break;

}

else

found = 0;

}

if (!found)

least[j] = -9999;

}

int min = 9999;

int repindex;

for (j = 0; j < nf; j++)

{

if (least[j] < min)

{

min = least[j];

repindex = j;

}

}

p[repindex] = in[i];

pgfaultcnt++;

dispPages();

}

else

printf("No page fault!");

}

dispPgFaultCnt();

}

void lfu()

{

int usedcnt[100];

int least, repin, sofarcnt = 0, bn;

initialize();

for (i = 0; i < nf; i++)

usedcnt[i] = 0;

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

{

printf("\n For %d :", in[i]);

if (isHit(in[i]))

{

int hitind = getHitIndex(in[i]);

usedcnt[hitind]++;

printf("No page fault!");

}

else

{

pgfaultcnt++;

if (bn < nf)

{

p[bn] = in[i];

usedcnt[bn] = usedcnt[bn] + 1;

bn++;

}

else

{

least = 9999;

for (k = 0; k < nf; k++)

if (usedcnt[k] < least)

{

least = usedcnt[k];

repin = k;

}

p[repin] = in[i];

sofarcnt = 0;

for (k = 0; k <= i; k++)

if (in[i] == in[k])

sofarcnt = sofarcnt + 1;

usedcnt[repin] = sofarcnt;

}

dispPages();

}

}

dispPgFaultCnt();

}

void secondchance()

{

int usedbit[50];

int victimptr = 0;

initialize();

for (i = 0; i < nf; i++)

usedbit[i] = 0;

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

{

printf("\nFor %d:", in[i]);

if (isHit(in[i]))

{

printf("No page fault!");

int hitindex = getHitIndex(in[i]);

if (usedbit[hitindex] == 0)

usedbit[hitindex] = 1;

}

else

{

pgfaultcnt++;

if (usedbit[victimptr] == 1)

{

do

{

usedbit[victimptr] = 0;

victimptr++;

if (victimptr == nf)

victimptr = 0;

}

while (usedbit[victimptr] != 0);

}

if (usedbit[victimptr] == 0)

{

p[victimptr] = in[i];

usedbit[victimptr] = 1;

victimptr++;

}

dispPages();

}

if (victimptr == nf)

victimptr = 0;

}

dispPgFaultCnt();

}

int main()

{

int choice;

while (1)

{

printf("\nPage Replacement Algorithms\n1.Enter data\n2.FIFO\n3.Optimal\n4.LRU\n5.LFU\n6.Second Chance\n7.Exit\nEnter your choice:");

scanf("%d", &choice);

switch (choice)

{

case 1 :

getData();

break;

case 2 :

fifo();

break;

case 3 :

optimal();

break;

case 4 :

lru();

break;

case 5 :

lfu();

break;

case 6 :

secondchance();

break;

default :

return 0;

break;

}

}

}

Output:-

