OS Assignment:-

STUDENT NAME : - RONIT MEHTA

STUDENT ID : - 11803363

EMAIL ADDRESS: - ronitmehta853@gmail.com

GIT HUB LINK : - https://github.com/ronit-lab/os-316assignment

TO see code :- https://github.com/ronit-lab/os-316assignment/blob/master/.gitignore

COURSE   TITLE: - OPERATING SYSTEM

COURSE CODE :- CSE-316

PROBLEM :-

In the given problem create a Process Id (PID) manager that help us to know about free PIDs and it also heck

that no two processes are having the same pid value Once a process terminates

the PID manager may provide its pid to new process.

Use the following variables or constants to know the range of possible pid values:

```
#define MIN PID 100

#define MAX PID 1000
```

You may use any data structure of your know to represent

process identifiers.

You can use the following functions :-

- int allocate map(void)—Creates and initializes a any data structure for which can use to represent pids;

returns—1 if fail, 1 if sucess

- int allocate pid(void)—provide pid to each unique process and returns a pid; returns— 1 which means cannot allocate else return 1 can allocate

a pid (all pids are in use)

- void release pid(int pid)—Releases a pid

Modify the given problem by making a multithreaded program that examine your solution.

You will create a number of threads—for example, 100—and each thread will allocate with a pid,which is unique

sleep for a random period of time, and then release the pid .The release pid is assign to new process.

SOLUTION :-

```
#include<stdio.h>
```

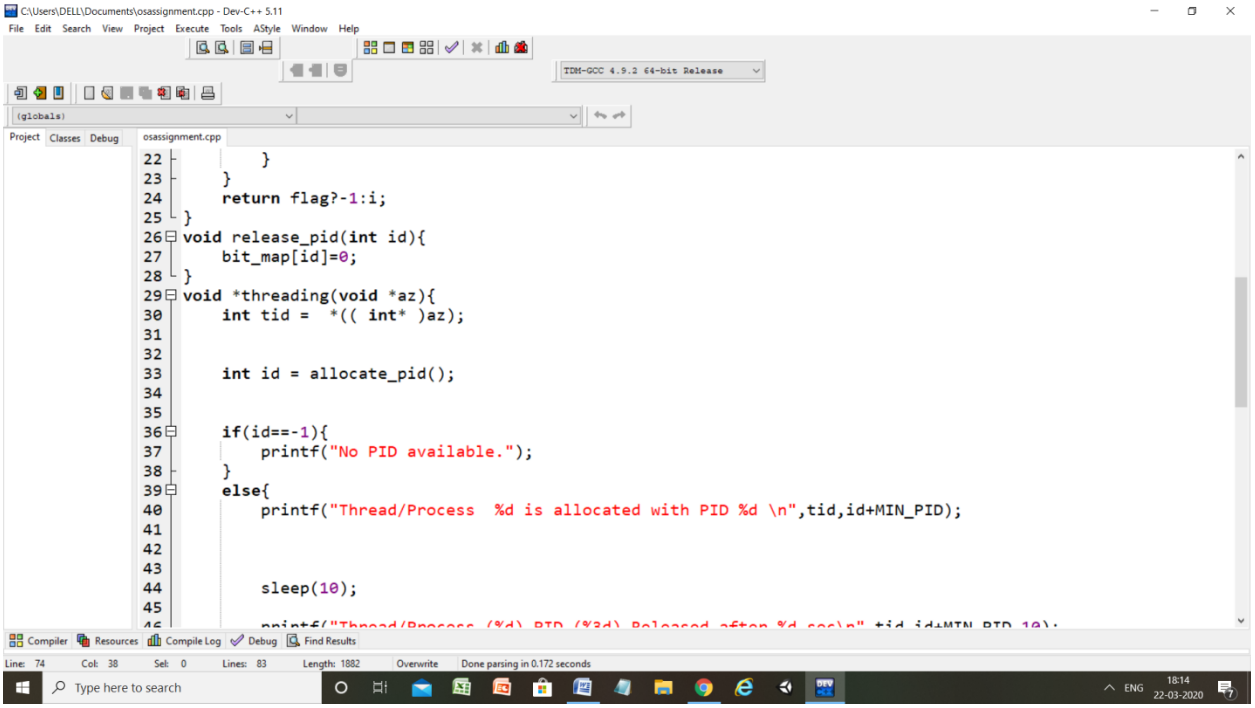
```
#include<stdlib.h>

#include<time.h>

#include<unistd.h>

#include<pthread.h>

#include<sys/types.h>

#define MIN_PID 100

#define MAX_PID 1000




int bit_map[MAX_PID-MIN_PID]={0};

int allocate_map()

{

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

bit_map[i]=0;

}


int allocate_pid(void){

    int i,flag=1;

    for(i=0; i<MAX_PID-MIN_PID; i++){

        if(bit_map[i]==0){


            bit_map[i]=1;


            flag=0;
```

```
      break;
    }
  }
  return flag?-1:i;
}
void release_pid(int id){
  bit_map[id]=0;
}
void *threading(void *az){
  int tid =  *(( int* )az);


  int id = allocate_pid();


  if(id==-1){
    printf("No PID available.");
  }
  else{
    printf("Thread  %d is allocated with PID %d \n",tid,id+MIN_PID);


    sleep(10);
    printf("Thread (%d) PID (%3d) Released after %d sec\n",tid,id+MIN_PID,10);


    release_pid(id);


  }
   printf("released PID is assign to new thread %d\n");
  pthread_exit(NULL);


}
void *processin(void *az){
  int tid =  *(( int* )az);



  int id = allocate_pid();



  if(id==-1){
    printf("No PID available.");
  }
  else{
    printf("Process  %d is allocated with PID %d \n",tid,id+MIN_PID);



    sleep(10);
```

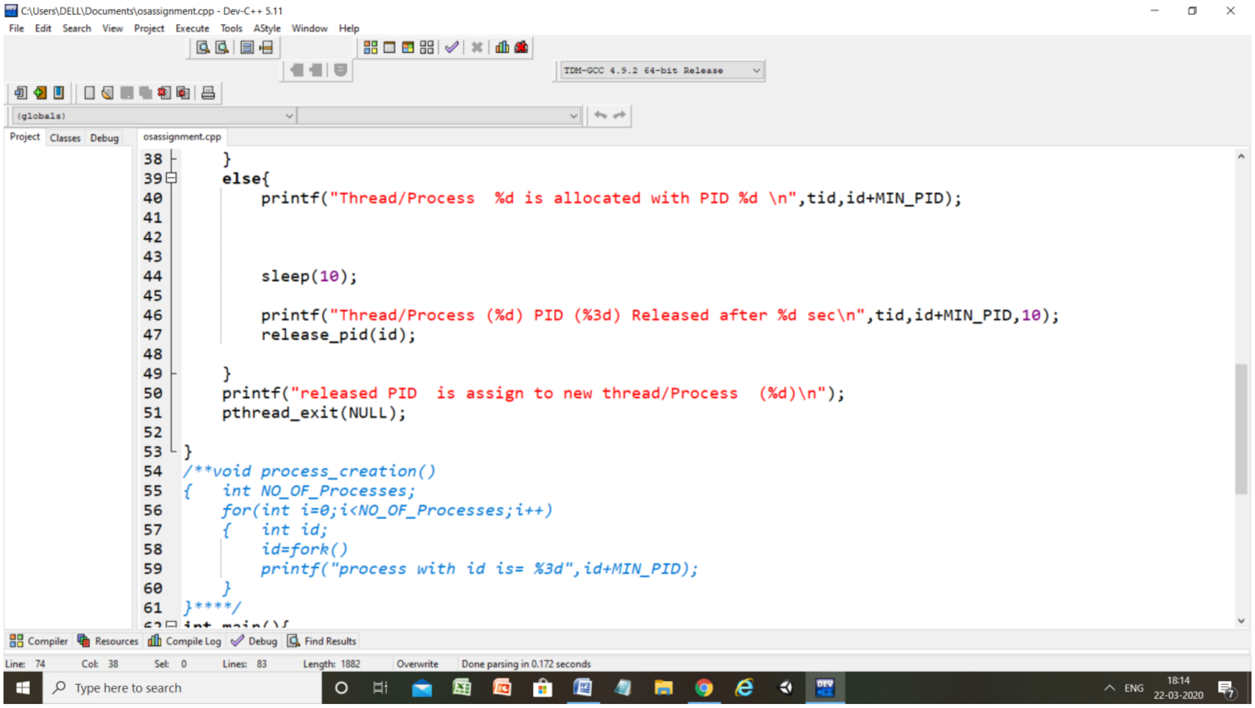
```
        printf("Process (%d) PID (%3d) Released after %d sec\n",tid,id+MIN_PID,10);
        release_pid(id);


    }
      printf("released PID is assign to new process %d\n");


    pthread_exit(NULL);


}


int main(){
            printf(">>>>>>>>>>>>>>>>>>>>>>WELCOME TO PID
MANAGER>>>>>>>>>>>>>>\n");
            allocate_pid();


      int num;
    printf("<<<<<<<<<<<<<<<<<<<<<<<ENTER YOUR
CHOICE>>>>>>>>>>>>>>>>>>>>\n");
    printf("1) FOR PROCESS CREATION \n");
    printf("2) FOR THREAD  CREATION \n");


    scanf("%d",&num);
    switch(num)
    {
            case 1:
                        printf("<<ITS PROCESS CREATION SECTION\n");
```

```
               int a;
                int tid;
                int id;
        int NO_OF_Processes;
        printf("enter the no of processes \n");
        scanf("%d",&NO_OF_Processes);
        for(int i=0;i<NO_OF_Processes;i++)
        {
                          pthread_t process[NO_OF_Processes];


      for(i=0; i<NO_OF_Processes; i++){
   if(pthread_create(&process[i],NULL,processin,(void*)&i))
      return -1*printf("Error in process %d creation !!!\n",i);
}


for(i=0; i<NO_OF_Processes; i++)
   pthread_join(process[i],NULL);


     }
               break;




      case 2:
```

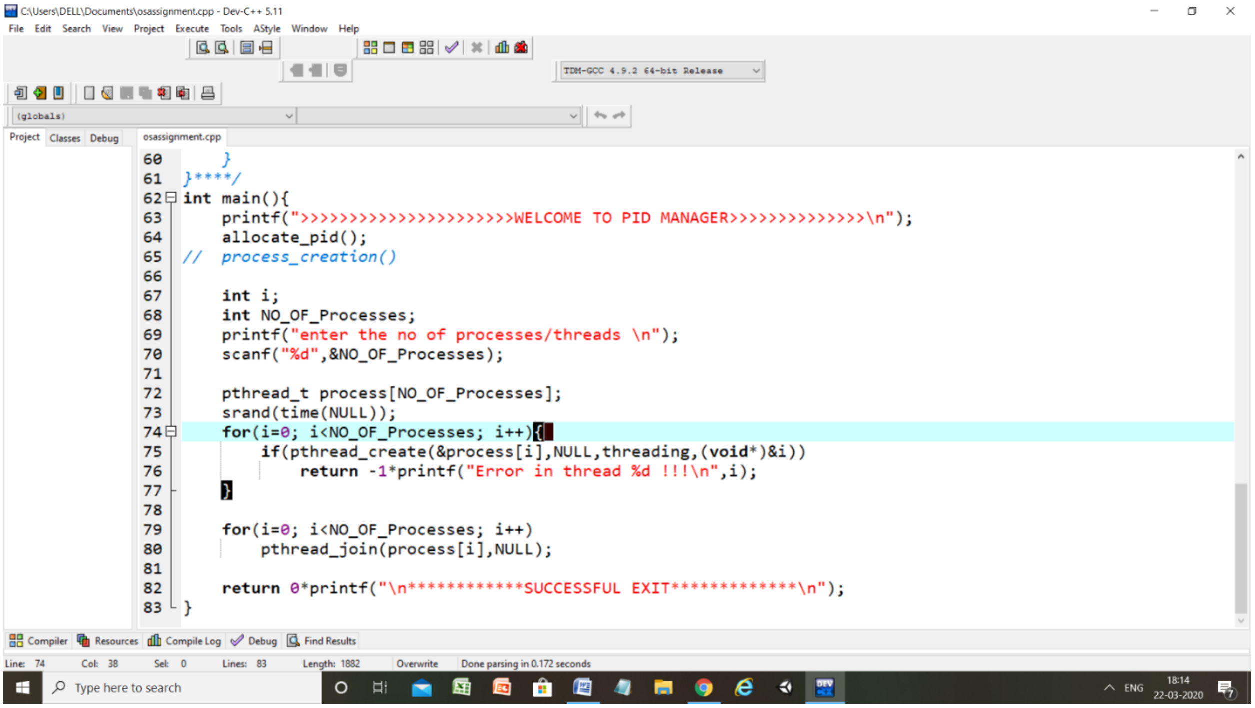
```
                        printf("<<ITS THREAD CREATION SECTION MAIN TESTCASE PRESS
100 \n");



   int i;

    int NO_OF_THREADS;

    printf("enter the no of threads \n");

    scanf("%d",&NO_OF_THREADS);


    pthread_t process[NO_OF_THREADS];


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

        if(pthread_create(&process[i],NULL,threading,(void*)&i))

            return -1*printf("Error in thread %d creation !!!\n",i);

    }


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

        pthread_join(process[i],NULL);


    return 0*printf("\n************PID DONE ITS WORK*************\n");

    break;



}

}
```


C:\Users\DELL\Documents\osassignment.cpp - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
TDM-GCC 4.9.2 64-bit Release
(globals)
Project Classes Debug
osassignment.cpp
Compiler Resources Compile Log Debug Find Results
Line: 74 Col: 38 Sel: 0 Lines: 83 Length: 1882 Overwrite Done parsing in 0.172 seconds
Type here to search
ENG 18:14 22-03-2020

```
        }
    }
    return flag?-1:i;
}
void release_pid(int id){
    bit_map[id]=0;
}
void *threading(void *az){
    int tid =  *(( int* )az);


    int id = allocate_pid();


    if(id==-1){
        printf("No PID available.");
    }
    else{
        printf("Thread/Process  %d is allocated with PID %d \n",tid,id+MIN_PID);



        sleep(10);

        printf("Thread/Process (%d) PID (%3d) Released after %d sec\n",tid,id+MIN_PID,10);
```


C:\Users\DELL\Documents\osassignment.cpp - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
TDM-GCC 4.9.2 64-bit Release
(globals)
Project Classes Debug
osassignment.cpp
Compiler Resources Compile Log Debug Find Results
Line: 74 Col: 38 Sel: 0 Lines: 83 Length: 1882 Overwrite Done parsing in 0.172 seconds
Type here to search
ENG 18:14 22-03-2020

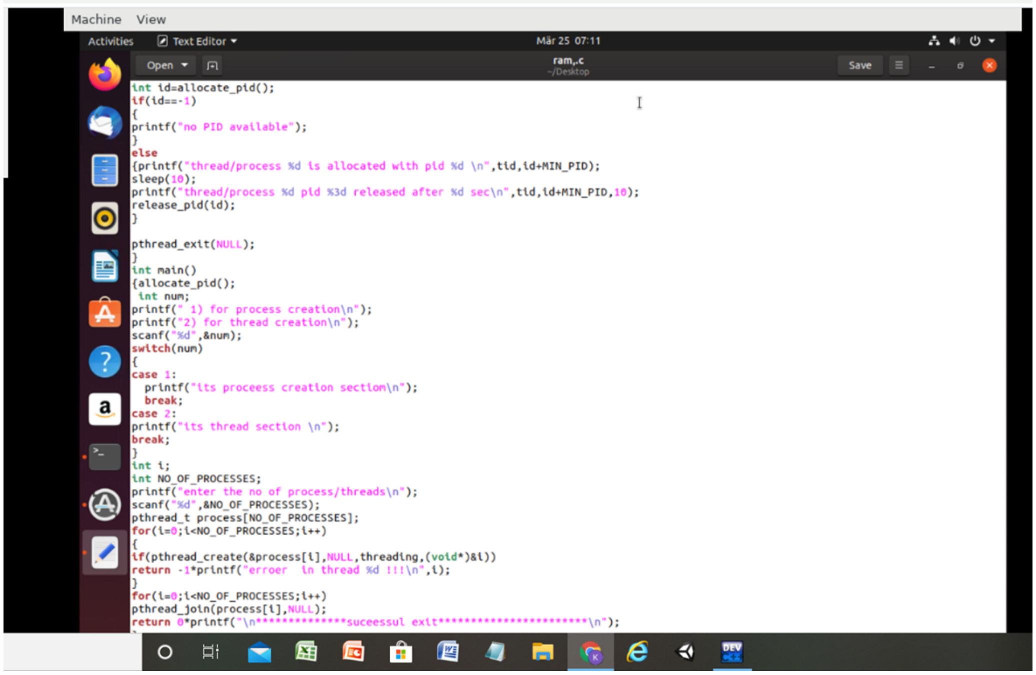
```
    }
    else{
        printf("Thread/Process  %d is allocated with PID %d \n",tid,id+MIN_PID);



        sleep(10);

        printf("Thread/Process (%d) PID (%3d) Released after %d sec\n",tid,id+MIN_PID,10);
        release_pid(id);

    }
    printf("released PID  is assign to new thread/Process  (%d)\n");
    pthread_exit(NULL);

}
/**void process_creation()
{   int NO_OF_Processes;
    for(int i=0;i<NO_OF_Processes;i++)
    {   int id;
        id=fork()
        printf("process with id is= %3d",id+MIN_PID);
    }
}****/
int main(){
```



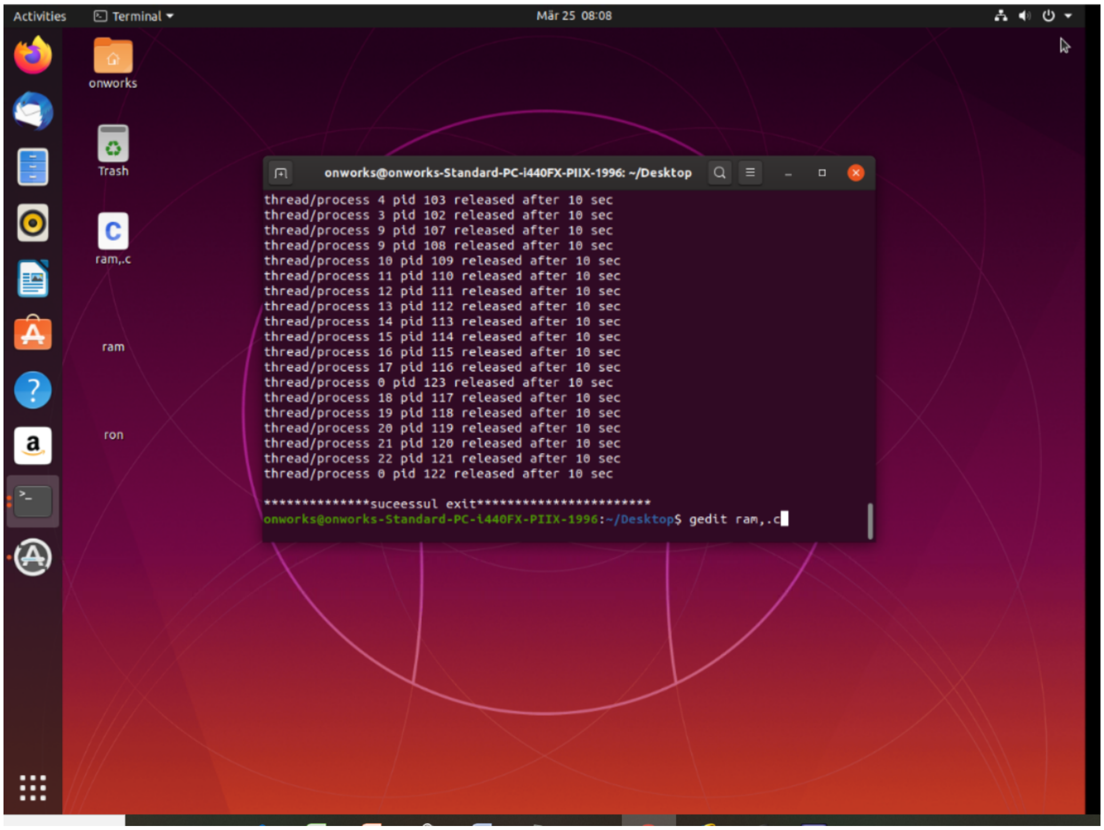
C:\Users\DELL\Documents\osassignment.cpp - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
TDM-GCC 4.9.2 64-bit Release
(globals)
Project Classes Debug
osassignment.cpp
Compiler Resources Compile Log Debug Find Results
Line: 74 Col: 38 Sel: 0 Lines: 83 Length: 1882 Overwrite Done parsing in 0.172 seconds
Type here to search
ENG 18:14 22-03-2020

```
    }
}****/
int main(){
    printf(">>>>>>>>>>>>>>>>>>>>>>WELCOME TO PID MANAGER>>>>>>>>>>>>>>\n");
    allocate_pid();
//  process_creation()

    int i;
    int NO_OF_Processes;
    printf("enter the no of processes/threads \n");
    scanf("%d",&NO_OF_Processes);

    pthread_t process[NO_OF_Processes];
    srand(time(NULL));
    for(i=0; i<NO_OF_Processes; i++){
        if(pthread_create(&process[i],NULL,threading,(void*)&i))
            return -1*printf("Error in thread %d !!!\n",i);
    }

    for(i=0; i<NO_OF_Processes; i++)
        pthread_join(process[i],NULL);

    return 0*printf("\n************SUCCESSFUL EXIT*************\n");
}
```

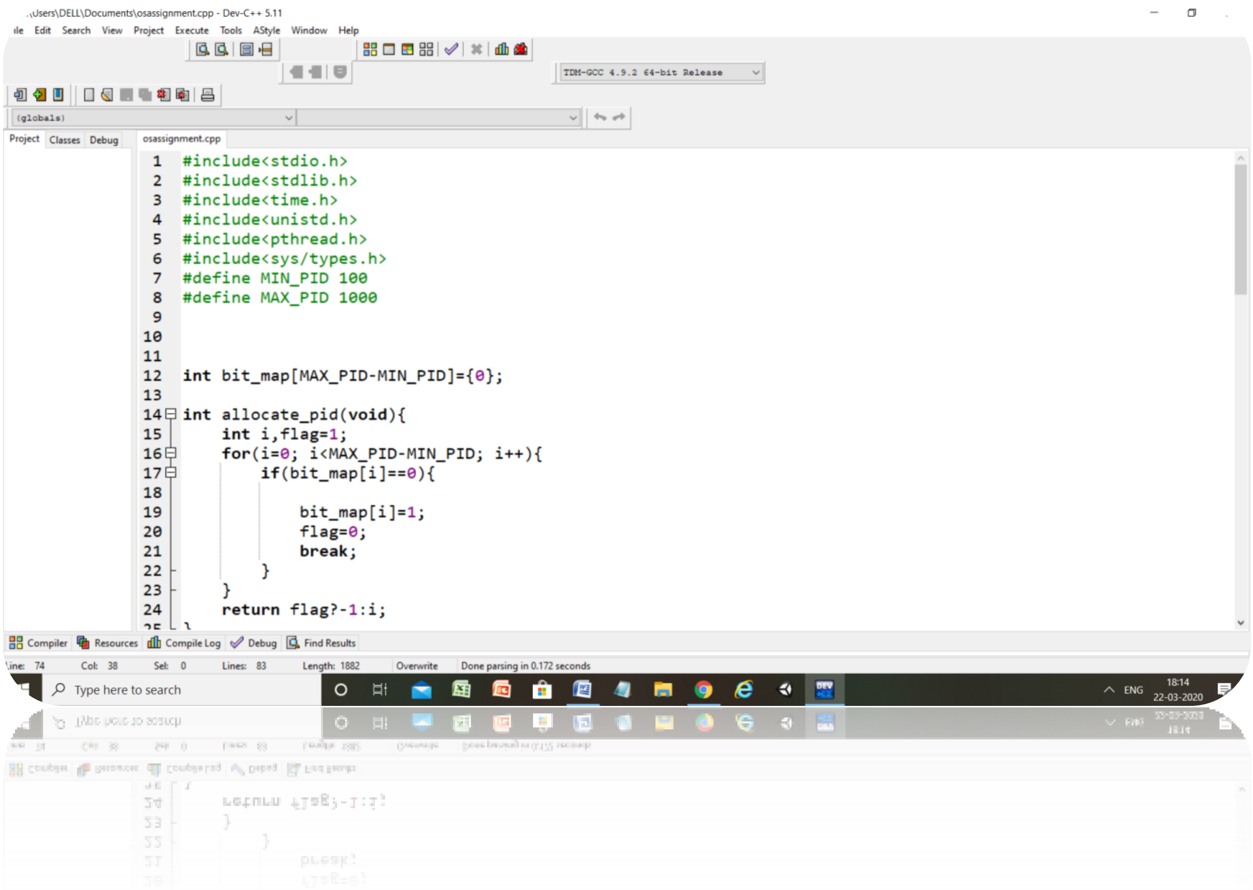
IMPLEMENTATION IN UBUNTO :-



```
int id=allocate_pid();
if(id==-1)
{
printf("no PID available");
}
else
{printf("thread/process %d is allocated with pid %d \n",tid,id+MIN_PID);
sleep(10);
printf("thread/process %d pid %3d released after %d sec\n",tid,id+MIN_PID,10);
release_pid(id);
}

pthread_exit(NULL);
}
int main()
{allocate_pid();
 int num;
printf(" 1) for process creation\n");
printf("2) for thread creation\n");
scanf("%d",&num);
switch(num)
{
case 1:
  printf("its proceess creation section\n");
  break;
case 2:
printf("its thread section \n");
break;
}
int i;
int NO_OF_PROCESSES;
printf("enter the no of process/threads\n");
scanf("%d",&NO_OF_PROCESSES);
pthread_t process[NO_OF_PROCESSES];
for(i=0;i<NO_OF_PROCESSES;i++)
{
if(pthread_create(&process[i],NULL,threading,(void*)&i))
return -1*printf("erroer  in thread %d !!!\n",i);
}
for(i=0;i<NO_OF_PROCESSES;i++)
pthread_join(process[i],NULL);
return 0*printf("\n*************suceessul exit**********************\n");
```



```
thread/process 4 pid 103 released after 10 sec
thread/process 3 pid 102 released after 10 sec
thread/process 9 pid 107 released after 10 sec
thread/process 9 pid 108 released after 10 sec
thread/process 10 pid 109 released after 10 sec
thread/process 11 pid 110 released after 10 sec
thread/process 12 pid 111 released after 10 sec
thread/process 13 pid 112 released after 10 sec
thread/process 14 pid 113 released after 10 sec
thread/process 15 pid 114 released after 10 sec
thread/process 16 pid 115 released after 10 sec
thread/process 17 pid 116 released after 10 sec
thread/process 0 pid 123 released after 10 sec
thread/process 18 pid 117 released after 10 sec
thread/process 19 pid 118 released after 10 sec
thread/process 20 pid 119 released after 10 sec
thread/process 21 pid 120 released after 10 sec
thread/process 22 pid 121 released after 10 sec
thread/process 0 pid 122 released after 10 sec

**************suceessul exit***********************
onworks@onworks-Standard-PC-i440FX-PIIX-1996:~/Desktop$ gedit ram,.c
```


..\Users\DELL\Documents\osassignment.cpp - Dev-C++ 5.11
ile Edit Search View Project Execute Tools AStyle Window Help
TDM-GCC 4.9.2 64-bit Release
(globals)
Project Classes Debug
osassignment.cpp
Compiler Resources Compile Log Debug Find Results
Type here to search
ENG
18:14
22-03-2020

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
#include<unistd.h>
#include<pthread.h>
#include<sys/types.h>
#define MIN_PID 100
#define MAX_PID 1000



int bit_map[MAX_PID-MIN_PID]={0};

int allocate_pid(void){
    int i,flag=1;
    for(i=0; i<MAX_PID-MIN_PID; i++){
        if(bit_map[i]==0){

            bit_map[i]=1;
            flag=0;
            break;
        }
    }
    return flag?-1:i;
}
```

```
Line: 74   Col: 38   Sel: 0   Lines: 83   Length: 1882   Overwrite   Done parsing in 0.172 seconds
```

DESCRITION:-

Use of each header file:

Include<stdio.h> :- This stdio.h header defines various variable types, macros as well as various functions for performing Input output operations.

Include<stdlib.h> :- This is library file for standard library

Include<pthread.h> :- header file used to use all the fuctions of pthread library

Include<time.h> :- the time.h header file contains definitions of function to get and manipulate date and time information.

Include <sys/types.h> :- defines various data types which are used in system source code.

In this project firstly two constants are declared which are used to assign the range of pid such that process identifier must lie between the given range .

# define MIN_PID used to assign minimum range from which the pid is assigned to process or threads

# define MAX_PID used to assign maximum range upto which pid is assigned.

Bit_map : preallocated bitmap function is used

Int bit_map[MAX_PID-MIN_PID]={0}

If(bit_map[i]==0) :- means pid available

Else : pid is in use

(If bit_map[i]==0) :- pid available and next line will be in critical section with a mutex lock whichever thread gets the mutex lock get the pid.

Int allocate_map(void) : this fuction is created to initialize bitmap function

Int allocate_pid(void): allocates and return the pid . it will check the condition if

It will return -1 it means all pid are in use and none is free to allocate and if it return 1 it means pid is available and next line should be in critical section with a mutex lock whichever thread gets the lock gets the pid // crtical section with mutex//

Void release_pid(int id) :- releases the pid means releases the mutex lock and come out of the critical section so that the released PID is assign to new process.

Process can be created with the help of fork() system call but here it is specify that int allocate pid(void) is used to provide id to all the processes.

Void *threading(void *az):- is used to cerate multithreaded system which assign PID to each thread int id=allocate_pid() will allocate unique pid to each and every process such that no process has same id. After assigning pid to every thread . each thread sleep for random amount of time using sleep function. And after that pid is released.

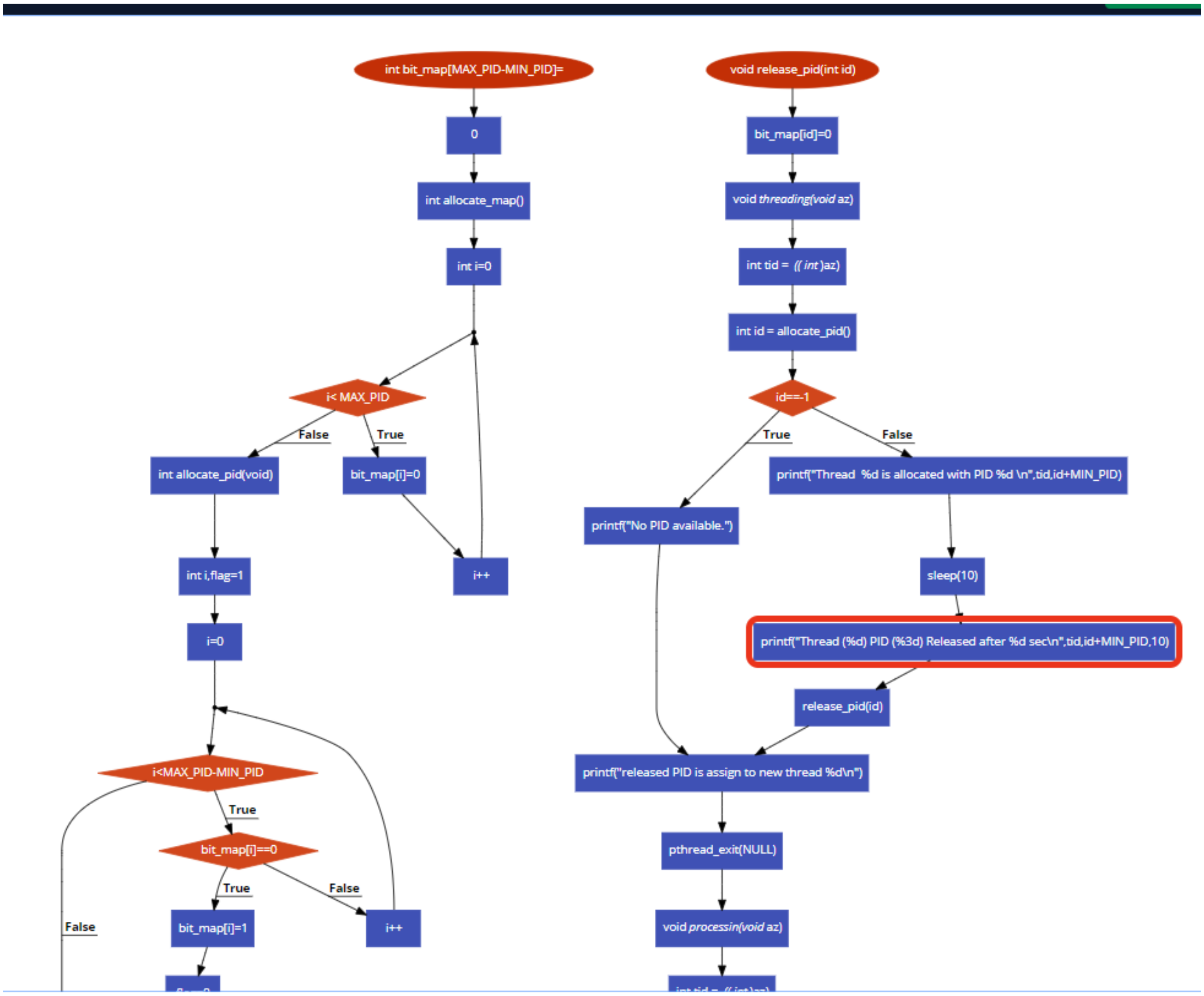
pthread_create ( ):- it is a function used to create a new thread with attributed specified by attribute within a processs . if attribute is null, the default attribute are used by pthhread_create function( )

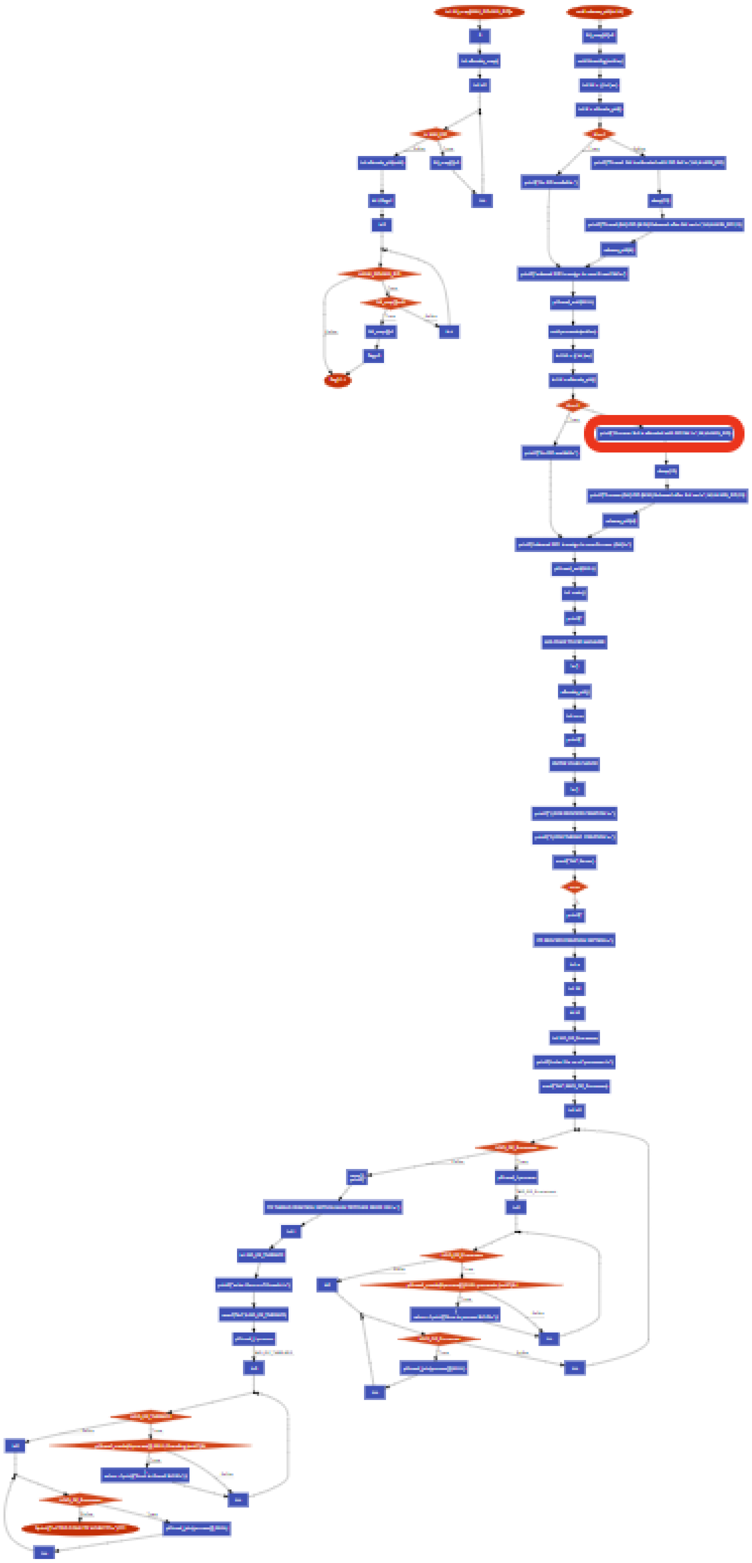
pthread_join():- This function is basically use to provide a simple mechanism which allow an application to wait for a thread to terminate.

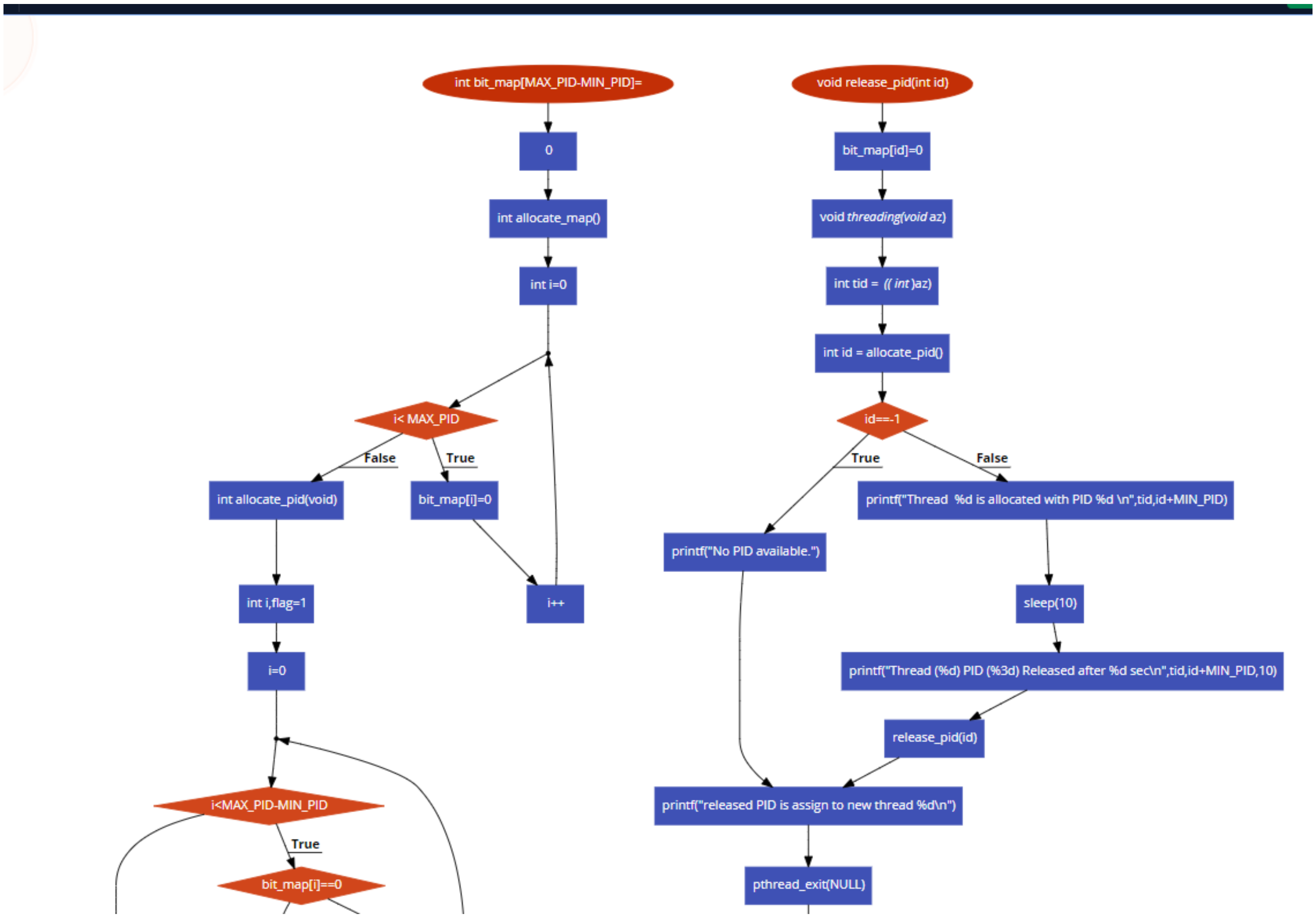
ALGORITHM USED:-FCFS (FIRST COME FIRST SERVE) :-It is the simplest form of process scheduling algorithm in which I/O requests are served according to the process arrival. The request arrives first will be accessed and served first .Since it follows the order of arrival,the process first allocate a unique pid and then other process according to their order of arrival. And process will release id also on the basis of their arrival.In his type of algorithm processes that requests CPU first get the allocation first.This is managed with the help of FIFO queue. For eg if we create Five processes p1,p2,p3,p4,p5 and so on each process is assign with PID according to their arrival in FIFO queue and released PID the same manner same is applicabe for thread also.

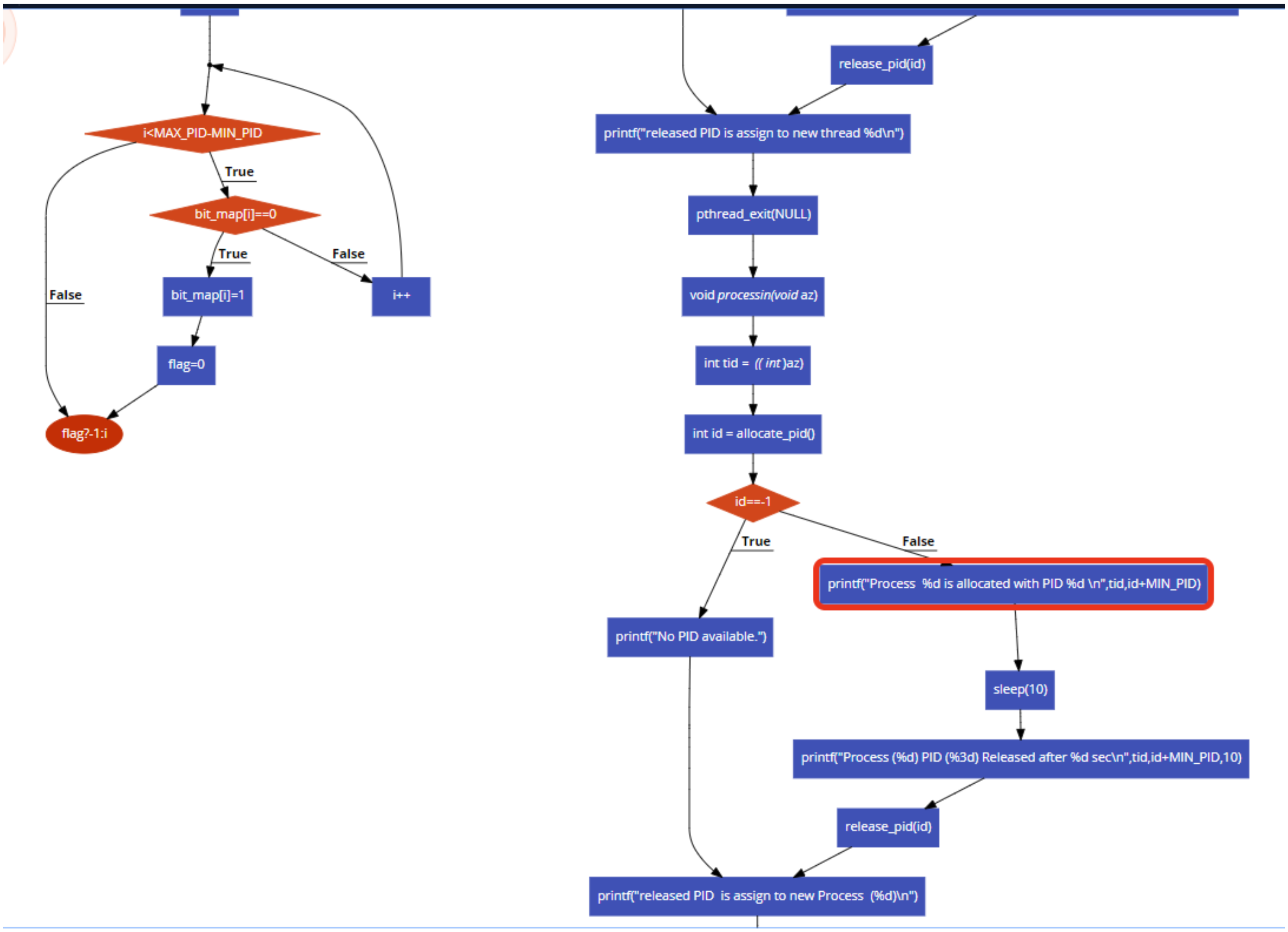
LOCKS :- A Lock is basically used to limit the access of resource when many threads wants to access that resources . it basically based on mutual exclusion which ever thread gets the mutex lock that thread enter into mutual exclusion and that time no other thread can enter into the critical section.

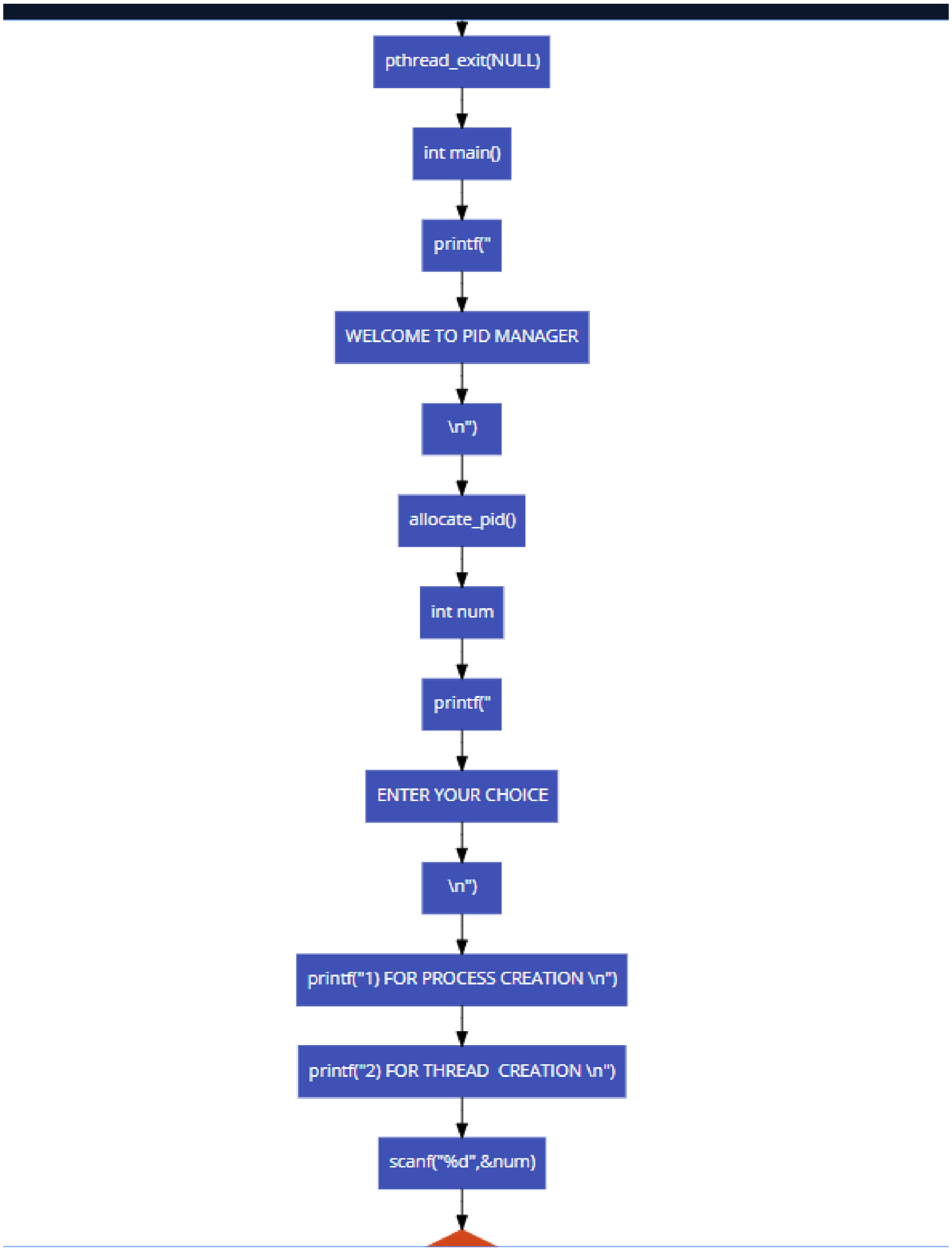
ALGORITHM – IN FROM OF FLOW CHART

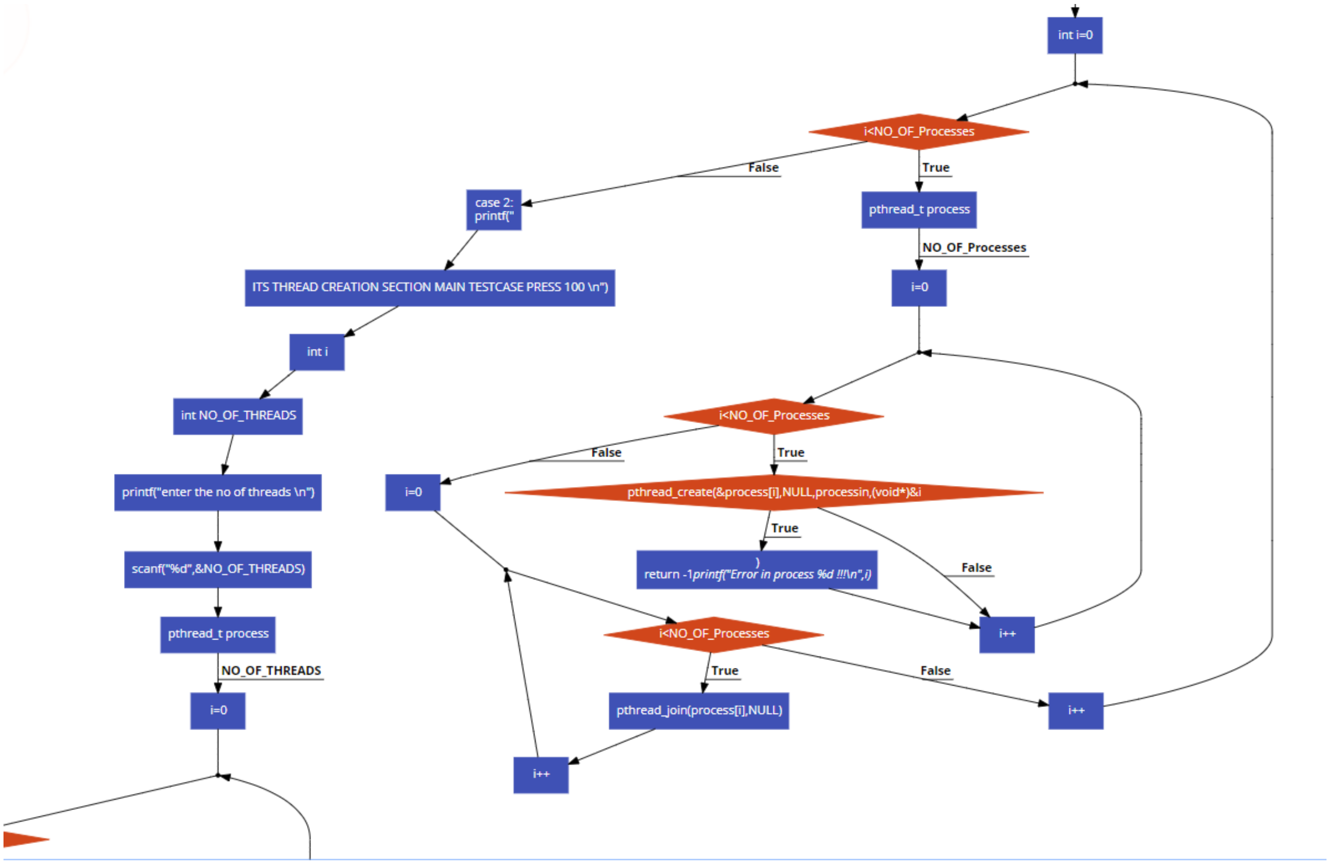

int bit_map[MAX_PID-MIN_PID]=
0
int allocate_map()
int i=0
i< MAX_PID
False
True
int allocate_pid(void)
bit_map[i]=0
i++
int i,flag=1
i=0
i<MAX_PID-MIN_PID
True
bit_map[i]==0
True
False
bit_map[i]=1
i++
False
void release_pid(int id)
bit_map[id]=0
void threading(void az)
int tid = ((int)az)
int id = allocate_pid()
id==-1
True
False
printf("Thread %d is allocated with PID %d \n",tid,id+MIN_PID)
printf("No PID available.")
sleep(10)
printf("Thread (%d) PID (%3d) Released after %d sec\n",tid,id+MIN_PID,10)
release_pid(id)
printf("released PID is assign to new thread %d\n")
pthread_exit(NULL)
void processin(void az)

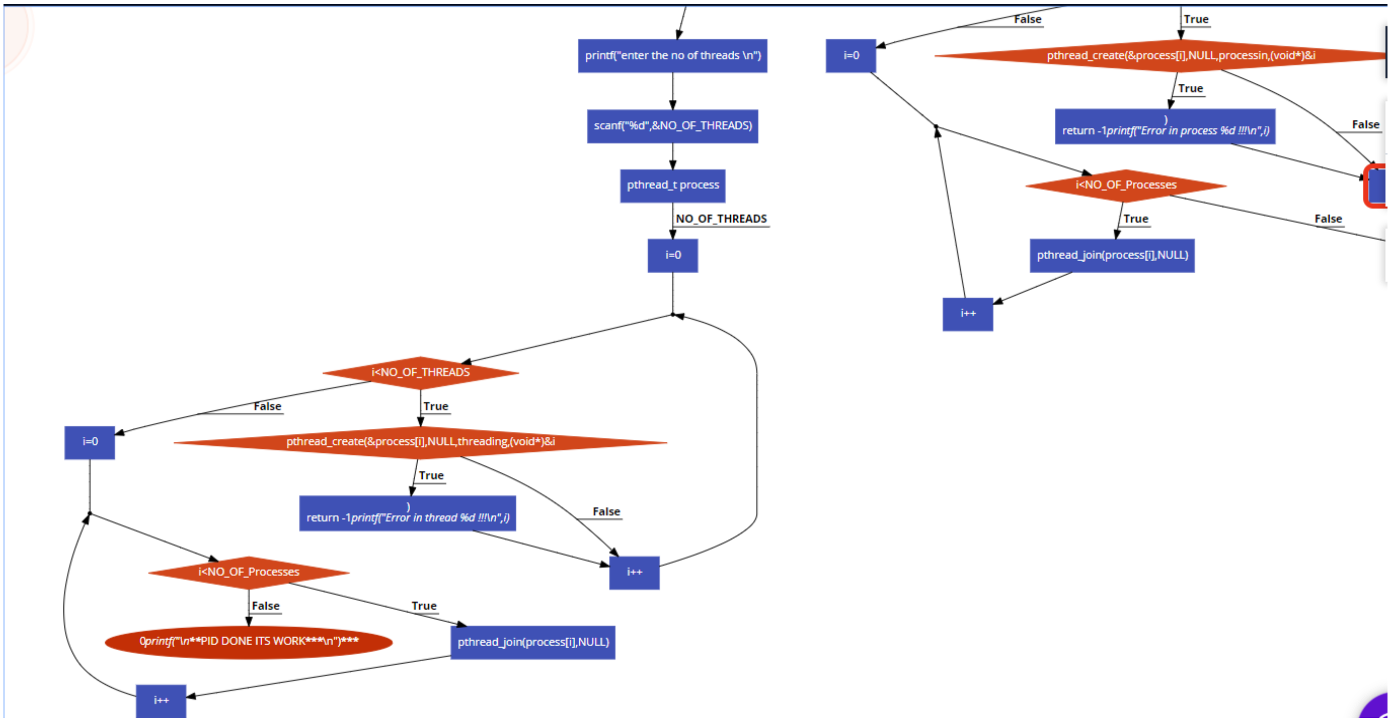



int bit_map[MAX_PID-MIN_PID]=
0
int allocate_map()
int i=0
i< MAX_PID
False
True
int allocate_pid(void)
bit_map[i]=0
i++
int i,flag=1
i=0
i<MAX_PID-MIN_PID
True
bit_map[i]==0
void release_pid(int id)
bit_map[id]=0
void threading(void az)
int tid = ((int)az)
int id = allocate_pid()
id==-1
True
False
printf("No PID available.")
printf("Thread %d is allocated with PID %d \n",tid,id+MIN_PID)
sleep(10)
printf("Thread (%d) PID (%3d) Released after %d sec\n",tid,id+MIN_PID,10)
release_pid(id)
printf("released PID is assign to new thread %d\n")
pthread_exit(NULL)


release_pid(id)
printf("released PID is assign to new thread %d\n")
pthread_exit(NULL)
void *processin(void az)*
int tid = ((*int*)az)
int id = allocate_pid()
id==-1
True
False
printf("Process %d is allocated with PID %d \n",tid,id+MIN_PID)
printf("No PID available.")
sleep(10)
printf("Process (%d) PID (%3d) Released after %d sec\n",tid,id+MIN_PID,10)
release_pid(id)
printf("released PID is assign to new Process (%d)\n")
i<MAX_PID-MIN_PID
True
False
bit_map[i]==0
True
False
bit_map[i]=1
i++
flag=0
flag?-1:i


pthread_exit(NULL)
int main()
printf("
WELCOME TO PID MANAGER
\n")
allocate_pid()
int num
printf("
ENTER YOUR CHOICE
\n")
printf("1) FOR PROCESS CREATION \n")
printf("2) FOR THREAD CREATION \n")
scanf("%d",&num)


int i=0
i<NO_OF_Processes
False
True
case 2:
printf("
pthread_t process
NO_OF_Processes
ITS THREAD CREATION SECTION MAIN TESTCASE PRESS 100 \n")
i=0
int i
int NO_OF_THREADS
i<NO_OF_Processes
False
True
printf("enter the no of threads \n")
i=0
pthread_create(&process[i],NULL,processin,(void*)&i
True
scanf("%d",&NO_OF_THREADS)
)
return -1printf("Error in process %d !!!\n",i)
False
pthread_t process
i<NO_OF_Processes
i++
NO_OF_THREADS
True
False
i=0
pthread_join(process[i],NULL)
i++
i++


False
True
printf("enter the no of threads \n")
i=0
pthread_create(&process[i],NULL,processin,(void*)&i
True
scanf("%d",&NO_OF_THREADS)
)
return -1printf("Error in process %d !!!\n",i)
False
pthread_t process
i<NO_OF_Processes
NO_OF_THREADS
True
False
i=0
pthread_join(process[i],NULL)
i++
i<NO_OF_THREADS
False
True
i=0
pthread_create(&process[i],NULL,threading,(void*)&i
True
False
)
return -1printf("Error in thread %d !!!\n",i)
i++
i<NO_OF_Processes
False
True
0printf("\n**PID DONE ITS WORK***\n")***
pthread_join(process[i],NULL)
i++

COMPLEXITY TABLE :-

CYCLOMATIC COMPLEXITY OF EACH FUNCTION USED IN CODE :-

FUNCTION NAME	CYCLOMATIC COMPLEXITY	NLOC
allocate_map( )	**2**	**5**
allocate_pid( )	**4**	**11**
release_pid( )	**1**	**3**
Threading	**2**	**15**
Main	**6**	**31**

TIME COMPLEXITY :- allocate map( ),allocate pid( ),release_pid( ) has complexity constant because it will depend upon NO_OF_PROCESSES WICH ARE CONSTANT whereas main( ) function has 0(n) complexity . **therefore overall complexity in o(n).**

FORMULA USED IN CALCULATION:- E-N+2P

E= NO OF EDGES

N= NO OF NODES

P= NO OF CONNECTED COMPONENTS IN GRAPH

CONSTRAINT GIVEN IN PROBLEM :- use the following function for obtaining and releasing a pid :

• int allocate map(void)—Creates and used for initialization of a data structure for representing pids;

Returns —1 if unsuccessful, 1 if successful

• int allocate pid(void)—Allocates and returns a pid; returns— 1 if unable to allocate

a pid (all pids are in use)

• void release pid(int pid)—Releases a pid

Modify the above problem by creating a multithreaded program that examine your solution.

You will create a number of threads—for example, 100—and each thread will allocate with a pid,which is unique sleep for a random period of time, and then release the pid .The release pid is assign to new process.

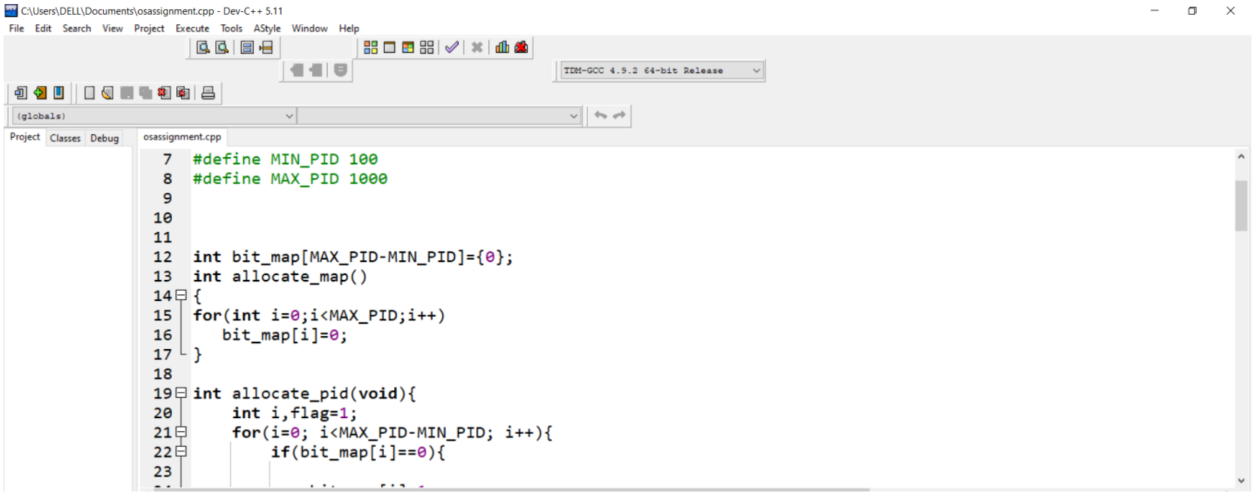
SOLUTION TO CONSTRAINT :-

• int allocate pid(void)—Allocates and returns a pid; returns— 1 if unable to allocate

a pid (all pids are in use)

CONSTRAINT :- • int allocate map(void)—Creates and initializes a data structure for representing pids;

Solution :- bitmap() is used to initialize and create pid's whose range lie between MIN_PID AND MAX_PID;

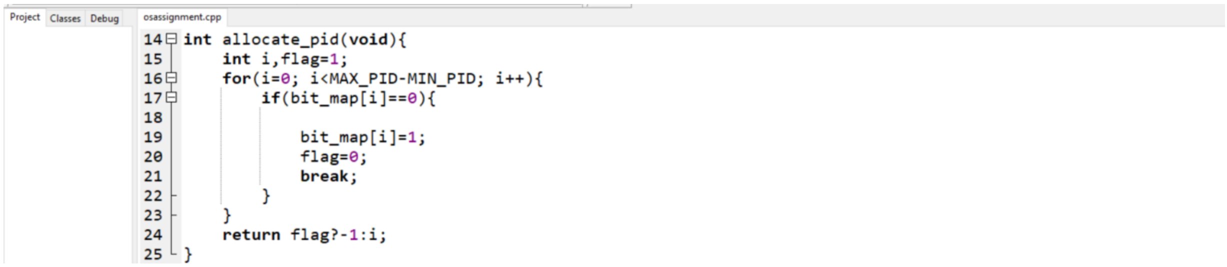

C:\Users\DELL\Documents\osassignment.cpp - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
TDM-GCC 4.9.2 64-bit Release
(globals)
Project Classes Debug
osassignment.cpp

```
#define MIN_PID 100
#define MAX_PID 1000


int bit_map[MAX_PID-MIN_PID]={0};
int allocate_map()
{
for(int i=0;i<MAX_PID;i++)
    bit_map[i]=0;
}

int allocate_pid(void){
    int i,flag=1;
    for(i=0; i<MAX_PID-MIN_PID; i++){
        if(bit_map[i]==0){

```

• int allocate pid(void)—Allocates and returns a pid; returns— 1 if unable to allocate

a pid (all pids are in use)

CODE SNIPPET :-

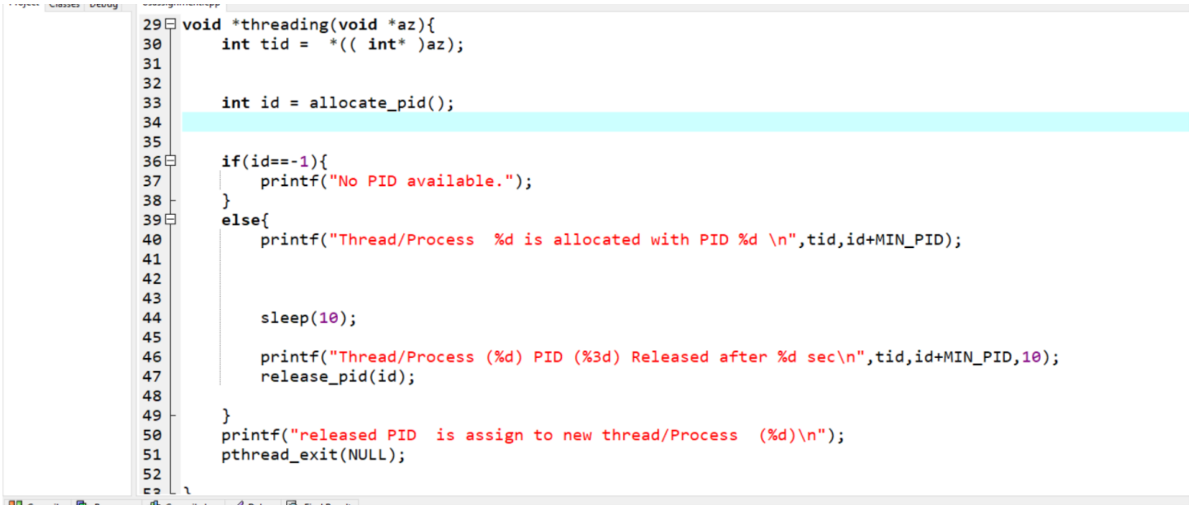


```
int allocate_pid(void){
    int i,flag=1;
    for(i=0; i<MAX_PID-MIN_PID; i++){
        if(bit_map[i]==0){

            bit_map[i]=1;
            flag=0;
            break;
        }
    }
    return flag?-1:i;
}
```

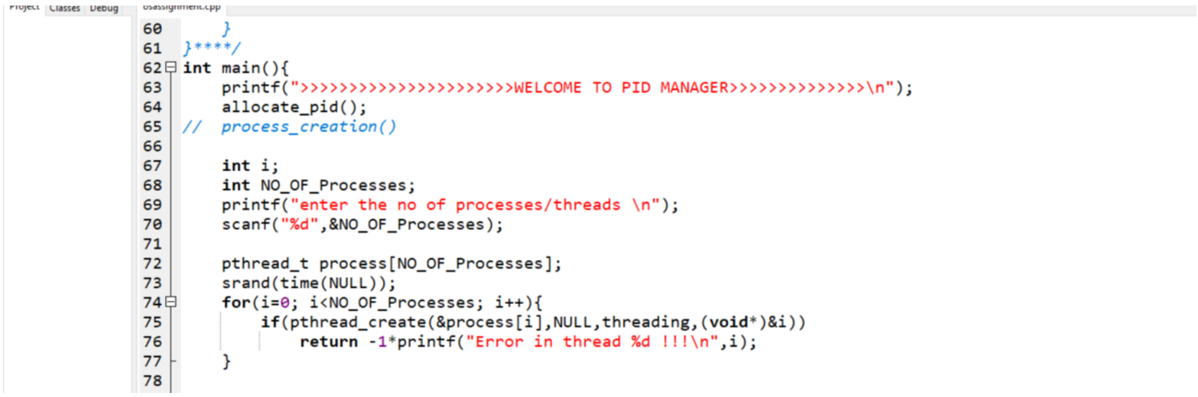
- void release pid(int pid)—Releases a pid

Modify the above problem by writing a multithreaded program that tests your solution.

You will create a number of threads—for example, 100—and each thread will request a pid, and sleep for random amount of time.



```
void *threading(void *az){
    int tid =  *(( int* )az);


    int id = allocate_pid();


    if(id==-1){
        printf("No PID available.");
    }
    else{
        printf("Thread/Process  %d is allocated with PID %d \n",tid,id+MIN_PID);



        sleep(10);

        printf("Thread/Process (%d) PID (%3d) Released after %d sec\n",tid,id+MIN_PID,10);
        release_pid(id);

    }
    printf("released PID  is assign to new thread/Process  (%d)\n");
    pthread_exit(NULL);

}
```

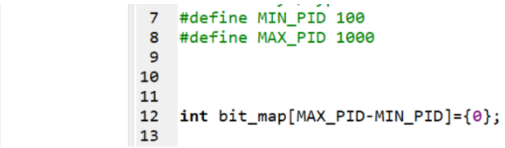


```
    }
}****/
int main(){
    printf(">>>>>>>>>>>>>>>>>>>>>>WELCOME TO PID MANAGER>>>>>>>>>>>>>>\n");
    allocate_pid();
//  process_creation()

    int i;
    int NO_OF_Processes;
    printf("enter the no of processes/threads \n");
    scanf("%d",&NO_OF_Processes);

    pthread_t process[NO_OF_Processes];
    srand(time(NULL));
    for(i=0; i<NO_OF_Processes; i++){
        if(pthread_create(&process[i],NULL,threading,(void*)&i))
            return -1*printf("Error in thread %d !!!\n",i);
    }

```

Use the following constants to identify the range of possible pid values:

#define MIN PID 100

#defin MAX PID 1000



```
#define MIN_PID 100
#define MAX_PID 1000


int bit_map[MAX_PID-MIN_PID]={0};

```

BOUNDARY CONDITIONS :- BOUNDARY CONDITION:-PID RANGE SHOULD LIE BETWEEN 100-1000 and each process must have different PID;

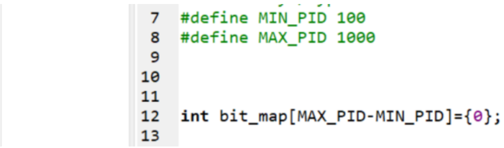
This code snippet ensure that PID must lie between 100-1000 and each process have different PID

We have used bit_map[] character array which is used to assign pids .This character array ensure that pid must lie between MAX_PID AND MIN_PID

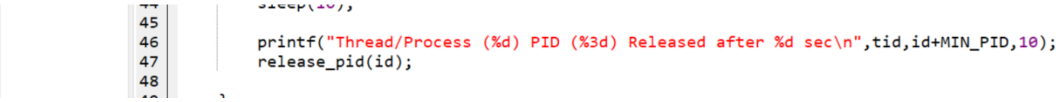
MAX_PID IS UPTO 1000

MIN_PID IS INITIALIZE 100

SO bit_map[] have range between 100-1000



```
#define MIN_PID 100
#define MAX_PID 1000


int bit_map[MAX_PID-MIN_PID]={0};

```



```
        sleep(10);

        printf("Thread/Process (%d) PID (%3d) Released after %d sec\n",tid,id+MIN_PID,10);
        release_pid(id);

    }
```

SECOND BOUNDARY CONDITION :- EACH PROCESS/THREAD SHOULD ASSIGGNED WITH UNIQUE PID.

We have create a function as

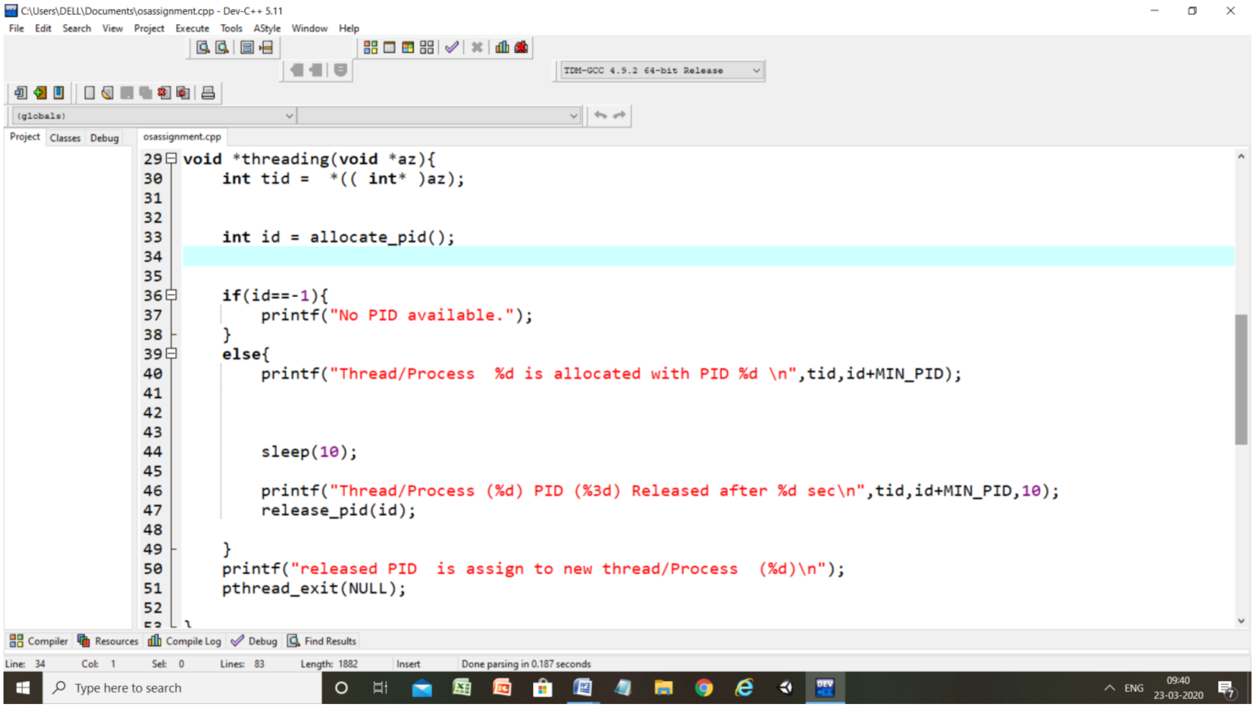
Void *threading(void *az)

{ int id=*((int *)az); // his will initialize id for hreads

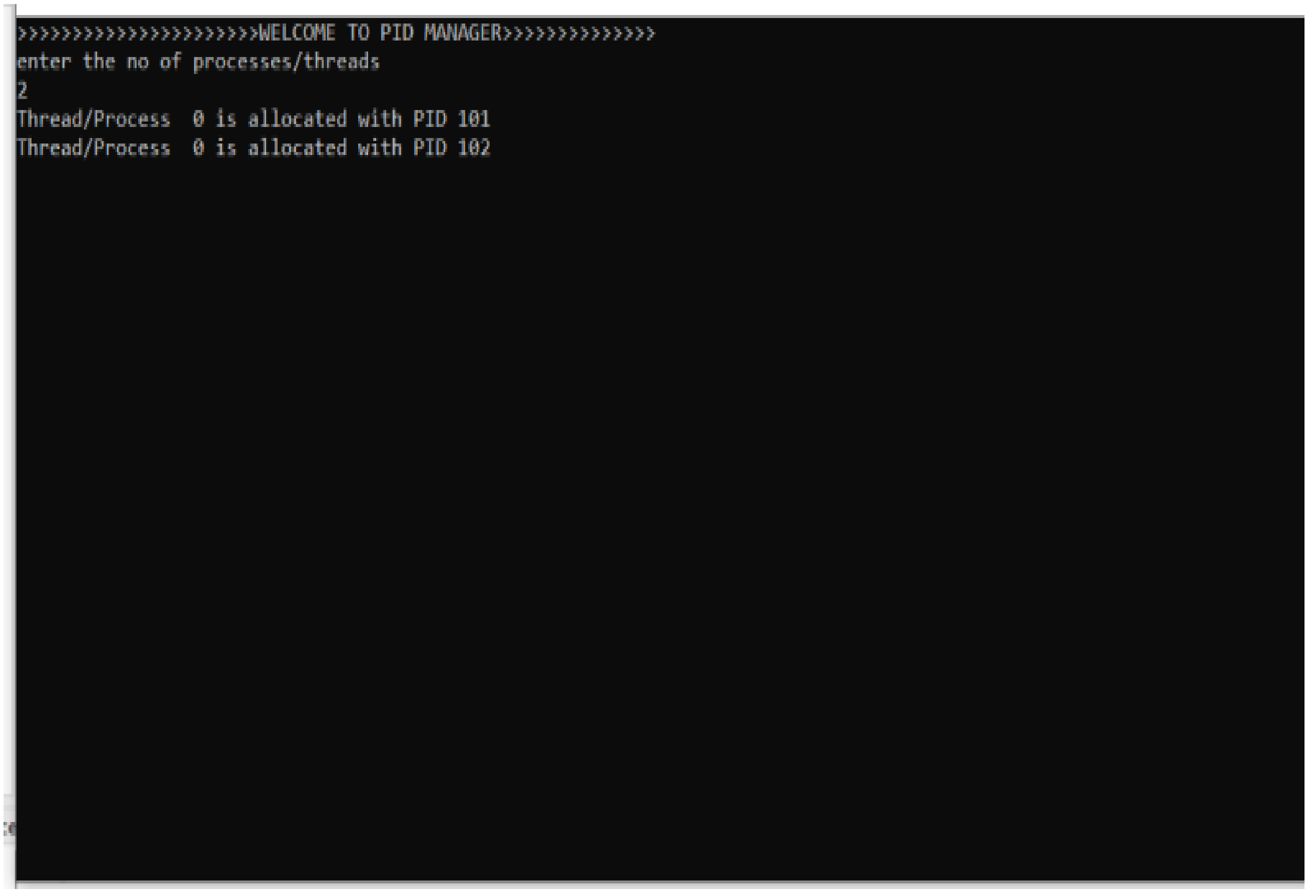
Int id= allocate_pid() // it will call he allocate_pid() method o allocate ids

Printf(hread/process %d is allocated with pid %d,tid,id+MIN_PID)

// this printf function allocate unique pid which are taken from allocate_pid() and added o min_pid,


C:\Users\DELL\Documents\osassignment.cpp - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
TDM-GCC 4.9.2 64-bit Release
(globals)
Project Classes Debug
osassignment.cpp
Compiler Resources Compile Log Debug Find Results
Line: 34 Col: 1 Sel: 0 Lines: 83 Length: 1882 Insert Done parsing in 0.187 seconds
Type here to search
ENG 09:40 23-03-2020

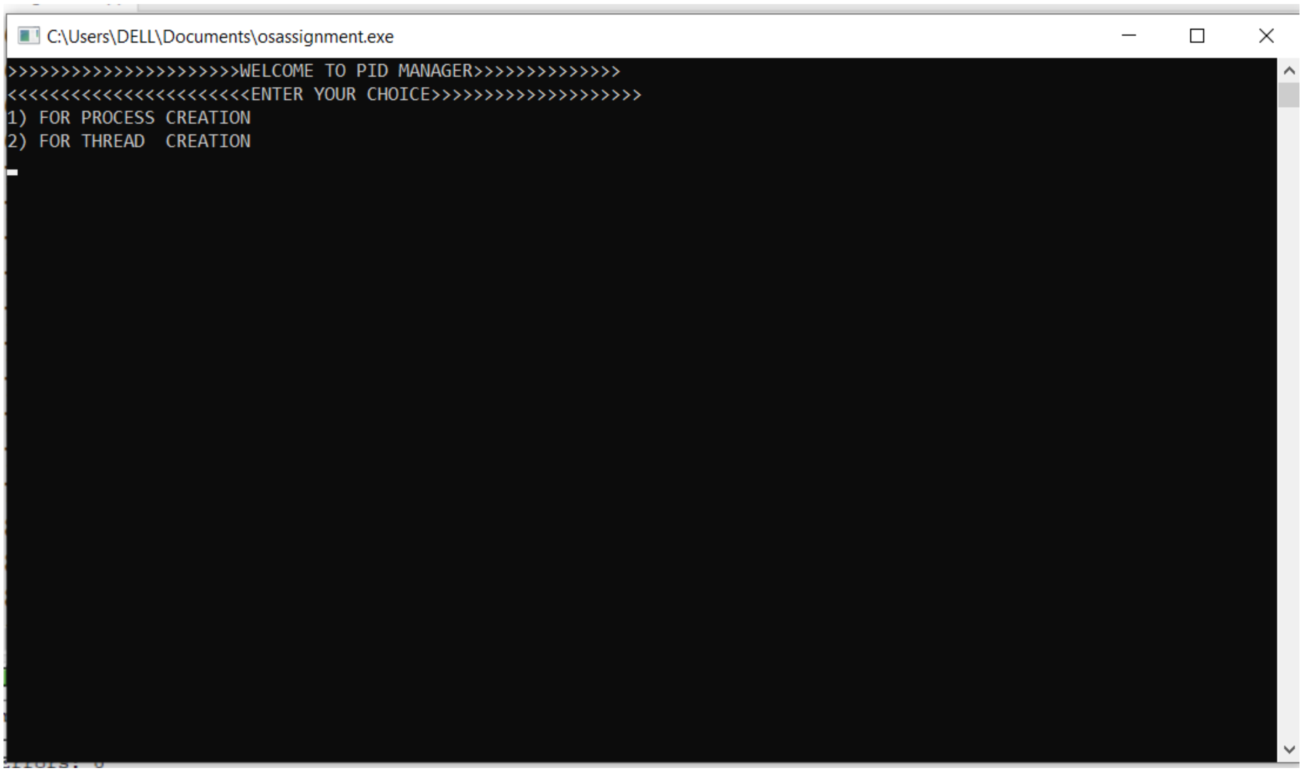
```
void *threading(void *az){
    int tid =  *(( int* )az);


    int id = allocate_pid();


    if(id==-1){
        printf("No PID available.");
    }
    else{
        printf("Thread/Process  %d is allocated with PID %d \n",tid,id+MIN_PID);



        sleep(10);

        printf("Thread/Process (%d) PID (%3d) Released after %d sec\n",tid,id+MIN_PID,10);
        release_pid(id);

    }
    printf("released PID  is assign to new thread/Process  (%d)\n");
    pthread_exit(NULL);

}
```



```
>>>>>>>>>>>>>>>>>>>>>>WELCOME TO PID MANAGER>>>>>>>>>>>>>>
enter the no of processes/threads
2
Thread/Process  0 is allocated with PID 101
Thread/Process  0 is allocated with PID 102
```

Additional idea used :- I have create a switch case which asked the user to create process or thread if user click 1 he   enter into process creation section and if he enter 2 he thread

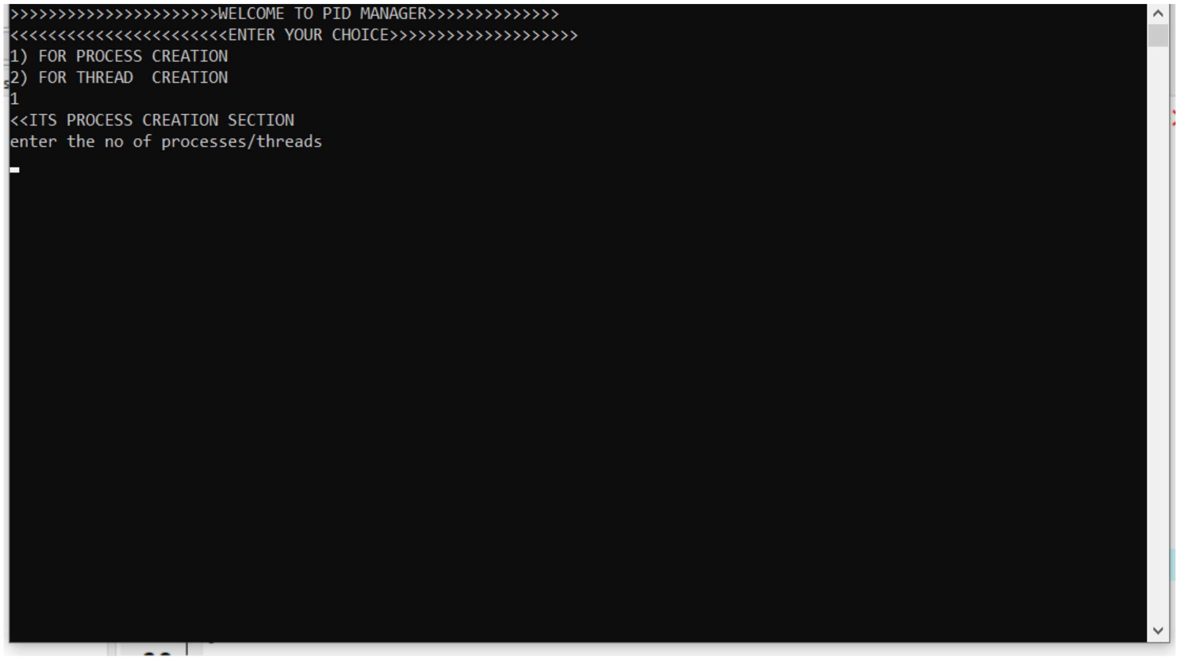
creation section.



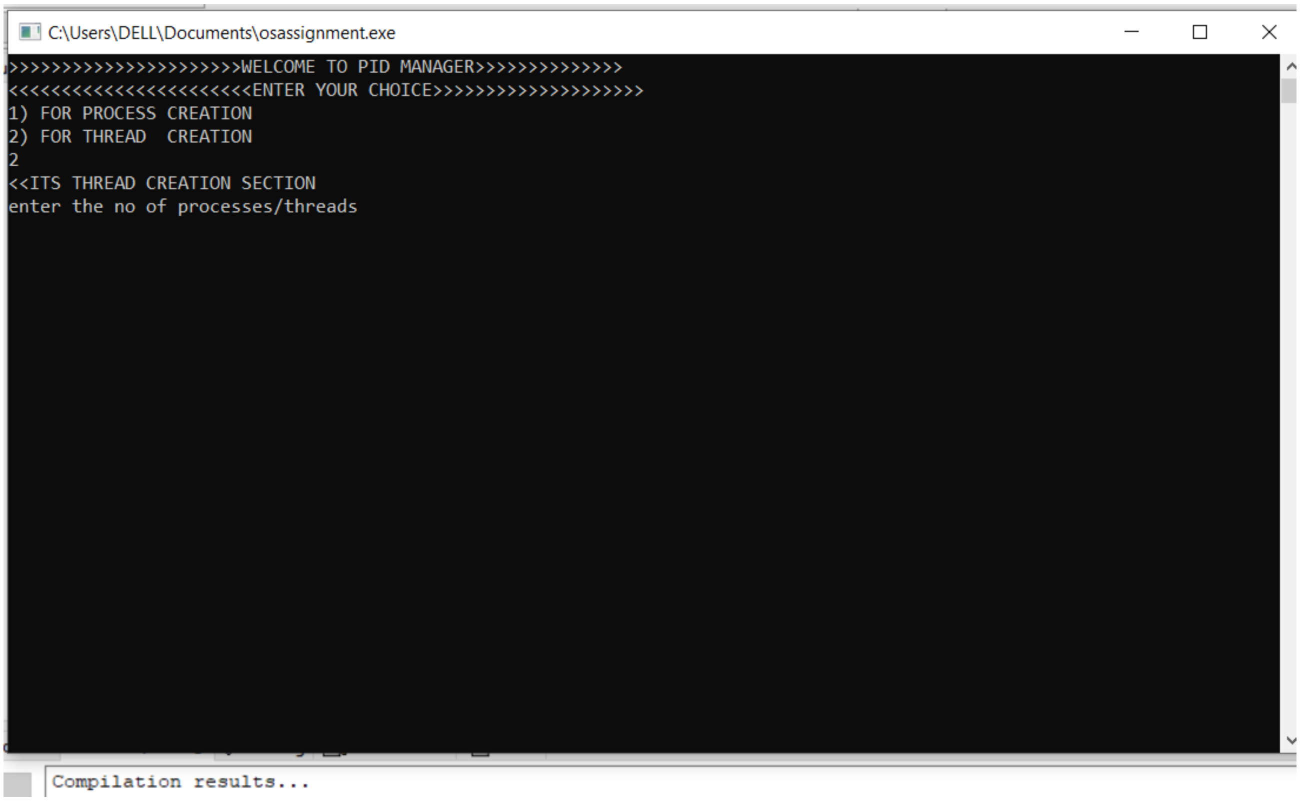
C:\Users\DELL\Documents\osassignment.exe

```
>>>>>>>>>>>>>>>>>>>>>>WELCOME TO PID MANAGER>>>>>>>>>>>>>>
<<<<<<<<<<<<<<<<<<<<<<<ENTER YOUR CHOICE>>>>>>>>>>>>>>>>>>>>
1) FOR PROCESS CREATION
2) FOR THREAD  CREATION
```

He enter 1 so he will enter in process creation section..and if he press 2 he will enter in hread creation section



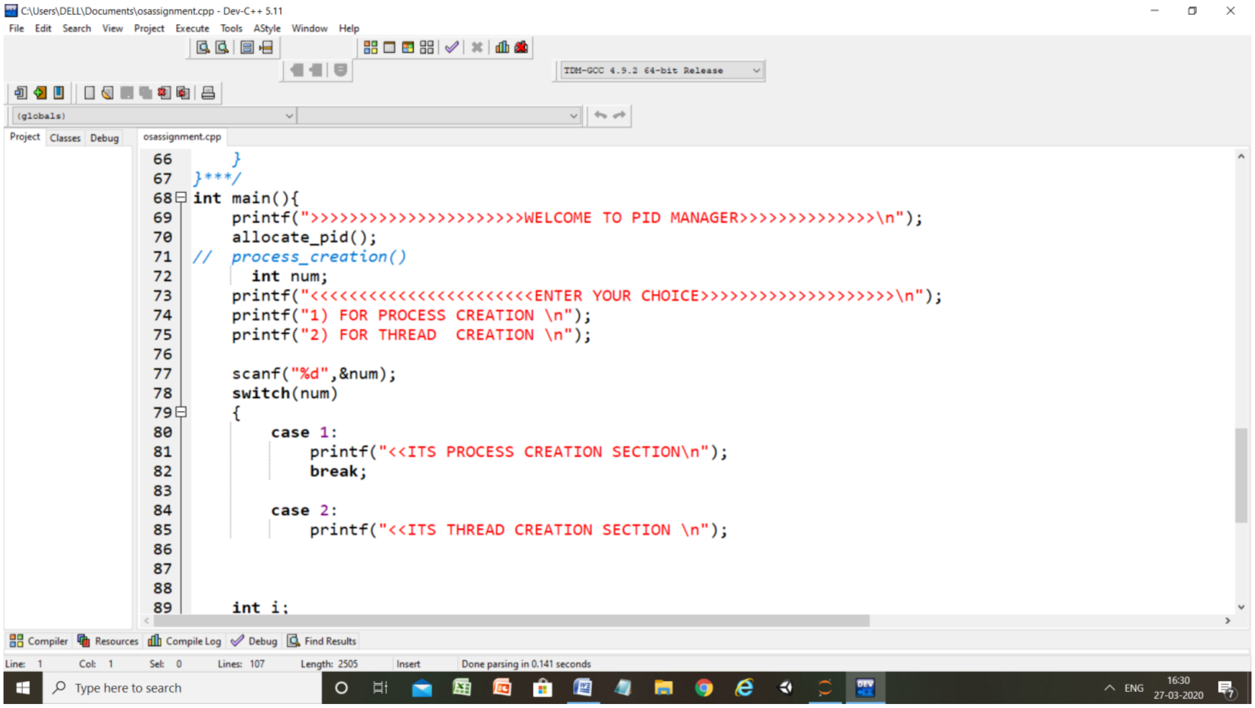
```
>>>>>>>>>>>>>>>>>>>>>>WELCOME TO PID MANAGER>>>>>>>>>>>>>>
<<<<<<<<<<<<<<<<<<<<<<<ENTER YOUR CHOICE>>>>>>>>>>>>>>>>>>>>
1) FOR PROCESS CREATION
2) FOR THREAD  CREATION
1
<<ITS PROCESS CREATION SECTION
enter the no of processes/threads
```



C:\Users\DELL\Documents\osassignment.exe

```
>>>>>>>>>>>>>>>>>>>>>>WELCOME TO PID MANAGER>>>>>>>>>>>>>>
<<<<<<<<<<<<<<<<<<<<<<<ENTER YOUR CHOICE>>>>>>>>>>>>>>>>>>>>
1) FOR PROCESS CREATION
2) FOR THREAD  CREATION
2
<<ITS THREAD CREATION SECTION
enter the no of processes/threads
```

```
Compilation results...
```



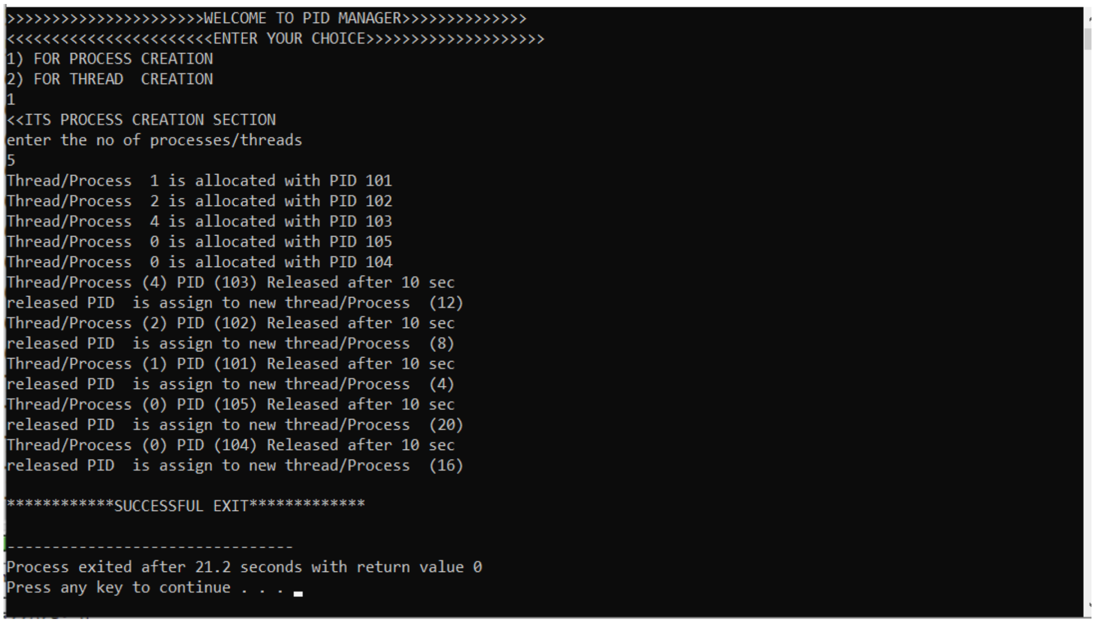
C:\Users\DELL\Documents\osassignment.cpp - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
TDM-GCC 4.9.2 64-bit Release
(globals)
Project Classes Debug
osassignment.cpp

```
        }
}***/
int main(){
    printf(">>>>>>>>>>>>>>>>>>>>>>WELCOME TO PID MANAGER>>>>>>>>>>>>>>\n");
    allocate_pid();
//  process_creation()
      int num;
    printf("<<<<<<<<<<<<<<<<<<<<<<<ENTER YOUR CHOICE>>>>>>>>>>>>>>>>>>>>\n");
    printf("1) FOR PROCESS CREATION \n");
    printf("2) FOR THREAD  CREATION \n");

    scanf("%d",&num);
    switch(num)
    {
        case 1:
            printf("<<ITS PROCESS CREATION SECTION\n");
            break;

        case 2:
            printf("<<ITS THREAD CREATION SECTION \n");



    int i;
```

Compiler Resources Compile Log Debug Find Results
Line: 1 Col: 1 Sel: 0 Lines: 107 Length: 2505 Insert Done parsing in 0.141 seconds
Type here to search
ENG 16:30 27-03-2020

--------------- : TESTCASES:-----------

TESTCASE 1 : Output of PID must lie between 100-1000 PID



```
>>>>>>>>>>>>>>>>>>>>>>WELCOME TO PID MANAGER>>>>>>>>>>>>>>
<<<<<<<<<<<<<<<<<<<<<<<ENTER YOUR CHOICE>>>>>>>>>>>>>>>>>>>>
1) FOR PROCESS CREATION
2) FOR THREAD  CREATION
1
<<ITS PROCESS CREATION SECTION
enter the no of processes/threads
5
Thread/Process  1 is allocated with PID 101
Thread/Process  2 is allocated with PID 102
Thread/Process  4 is allocated with PID 103
Thread/Process  0 is allocated with PID 105
Thread/Process  0 is allocated with PID 104
Thread/Process (4) PID (103) Released after 10 sec
released PID  is assign to new thread/Process  (12)
Thread/Process (2) PID (102) Released after 10 sec
released PID  is assign to new thread/Process  (8)
Thread/Process (1) PID (101) Released after 10 sec
released PID  is assign to new thread/Process  (4)
Thread/Process (0) PID (105) Released after 10 sec
released PID  is assign to new thread/Process  (20)
Thread/Process (0) PID (104) Released after 10 sec
released PID  is assign to new thread/Process  (16)

************SUCCESSFUL EXIT*************

--------------------------------
Process exited after 21.2 seconds with return value 0
Press any key to continue . . .
```

TEST CASE :- PID SHOULD BE UNIQUE EACH PID SHOULD HAVE DIFFERENT VALUE :- for eg :- here are 5 processess :-

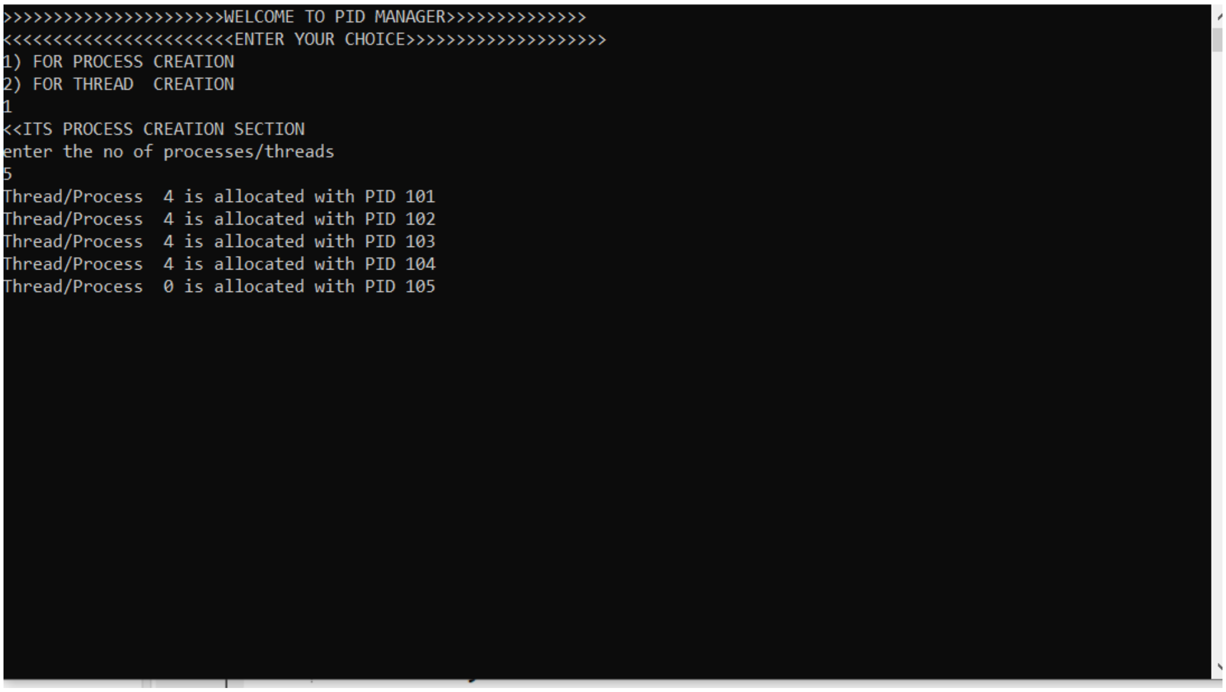
4 with PID 101

4 with PID 102

4 with PID 103

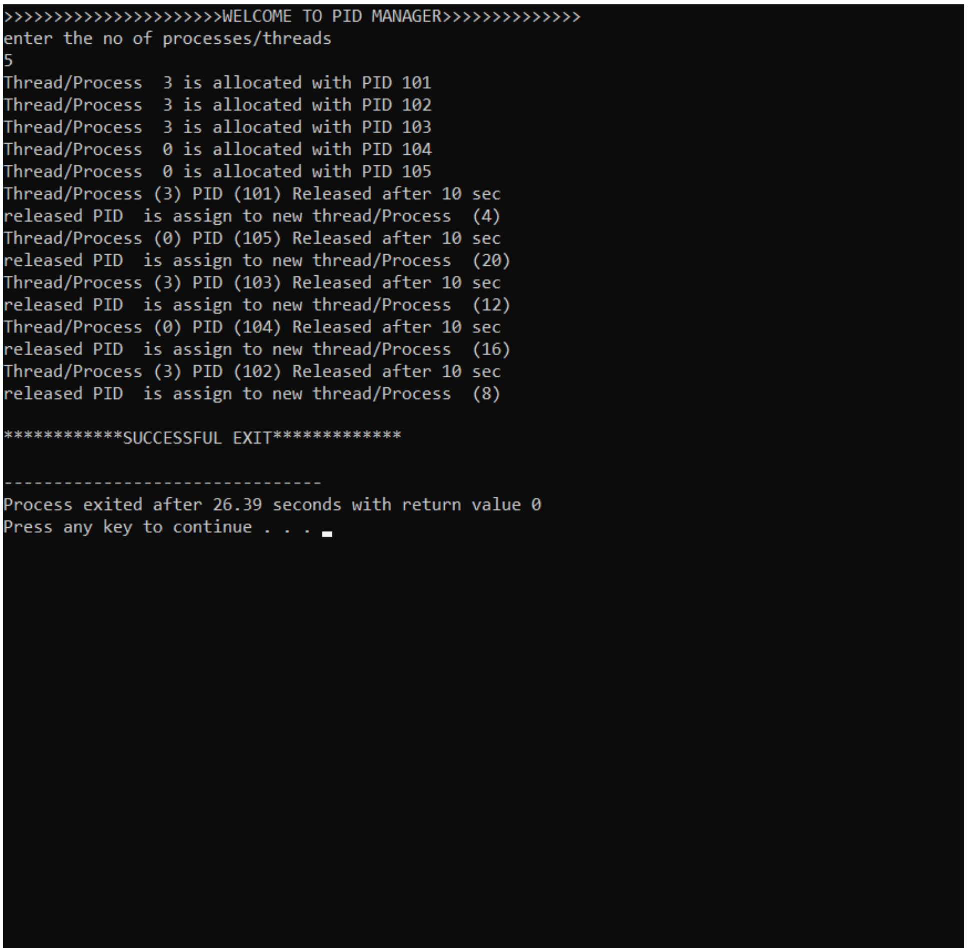
4 with PID 104

0 with PID 105



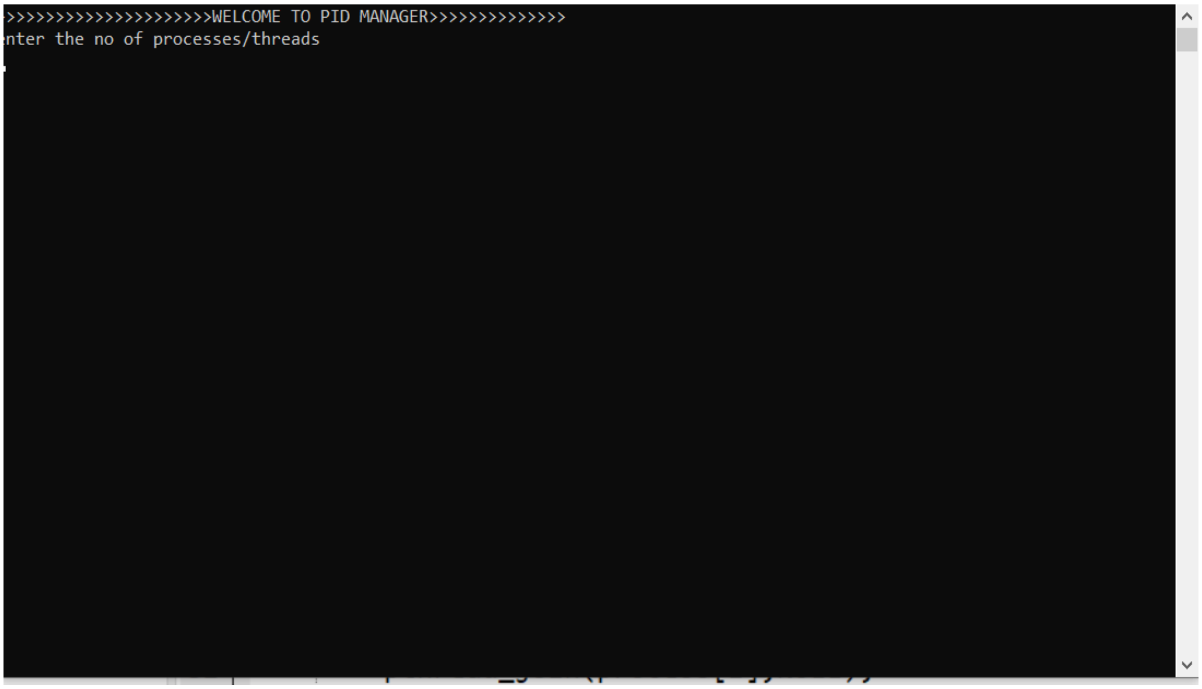
```
>>>>>>>>>>>>>>>>>>>>>>WELCOME TO PID MANAGER>>>>>>>>>>>>>>
<<<<<<<<<<<<<<<<<<<<<<<ENTER YOUR CHOICE>>>>>>>>>>>>>>>>>>>
1) FOR PROCESS CREATION
2) FOR THREAD  CREATION
1
<<ITS PROCESS CREATION SECTION
enter the no of processes/threads
5
Thread/Process  4 is allocated with PID 101
Thread/Process  4 is allocated with PID 102
Thread/Process  4 is allocated with PID 103
Thread/Process  4 is allocated with PID 104
Thread/Process  0 is allocated with PID 105
```

TESTCASE 3 :- Create 100 hreads allocate unique pid o each and hen sleep for random amount of ime and relesed its pid . Released pid is assigned o new process/thread.



```
>>>>>>>>>>>>>>>>>>>>>>WELCOME TO PID MANAGER>>>>>>>>>>>>>>
enter the no of processes/threads
5
Thread/Process  3 is allocated with PID 101
Thread/Process  3 is allocated with PID 102
Thread/Process  3 is allocated with PID 103
Thread/Process  0 is allocated with PID 104
Thread/Process  0 is allocated with PID 105
Thread/Process (3) PID (101) Released after 10 sec
released PID  is assign to new thread/Process  (4)
Thread/Process (0) PID (105) Released after 10 sec
released PID  is assign to new thread/Process  (20)
Thread/Process (3) PID (103) Released after 10 sec
released PID  is assign to new thread/Process  (12)
Thread/Process (0) PID (104) Released after 10 sec
released PID  is assign to new thread/Process  (16)
Thread/Process (3) PID (102) Released after 10 sec
released PID  is assign to new thread/Process  (8)

************SUCCESSFUL EXIT*************

--------------------------------
Process exited after 26.39 seconds with return value 0
Press any key to continue . . .
```

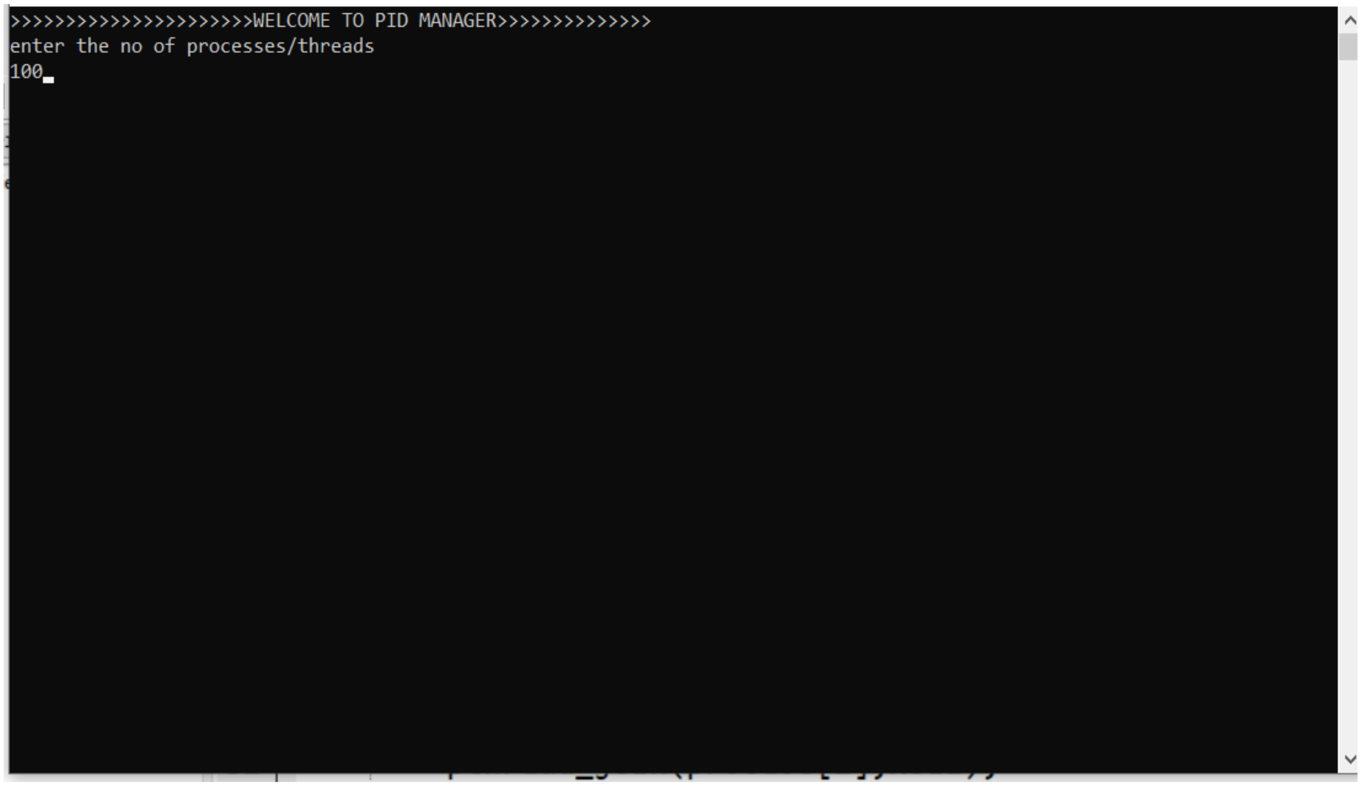
TESTCASE :-



```
>>>>>>>>>>>>>>>>>>>>>WELCOME TO PID MANAGER>>>>>>>>>>>>>>
nter the no of processes/threads
```

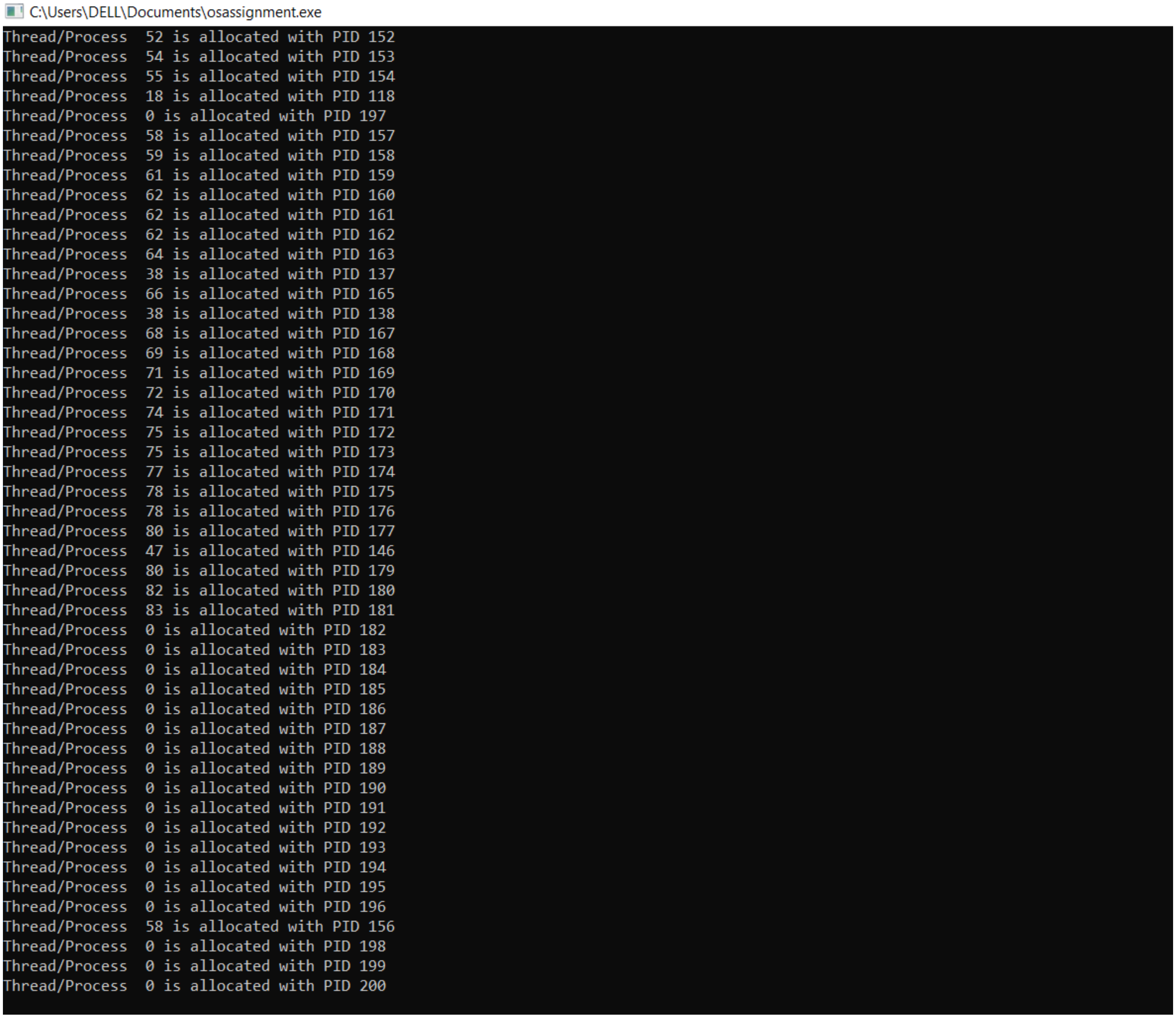
…..

100 is given as input



```
>>>>>>>>>>>>>>>>>>>>>>WELCOME TO PID MANAGER>>>>>>>>>>>>>>
enter the no of processes/threads
100
```

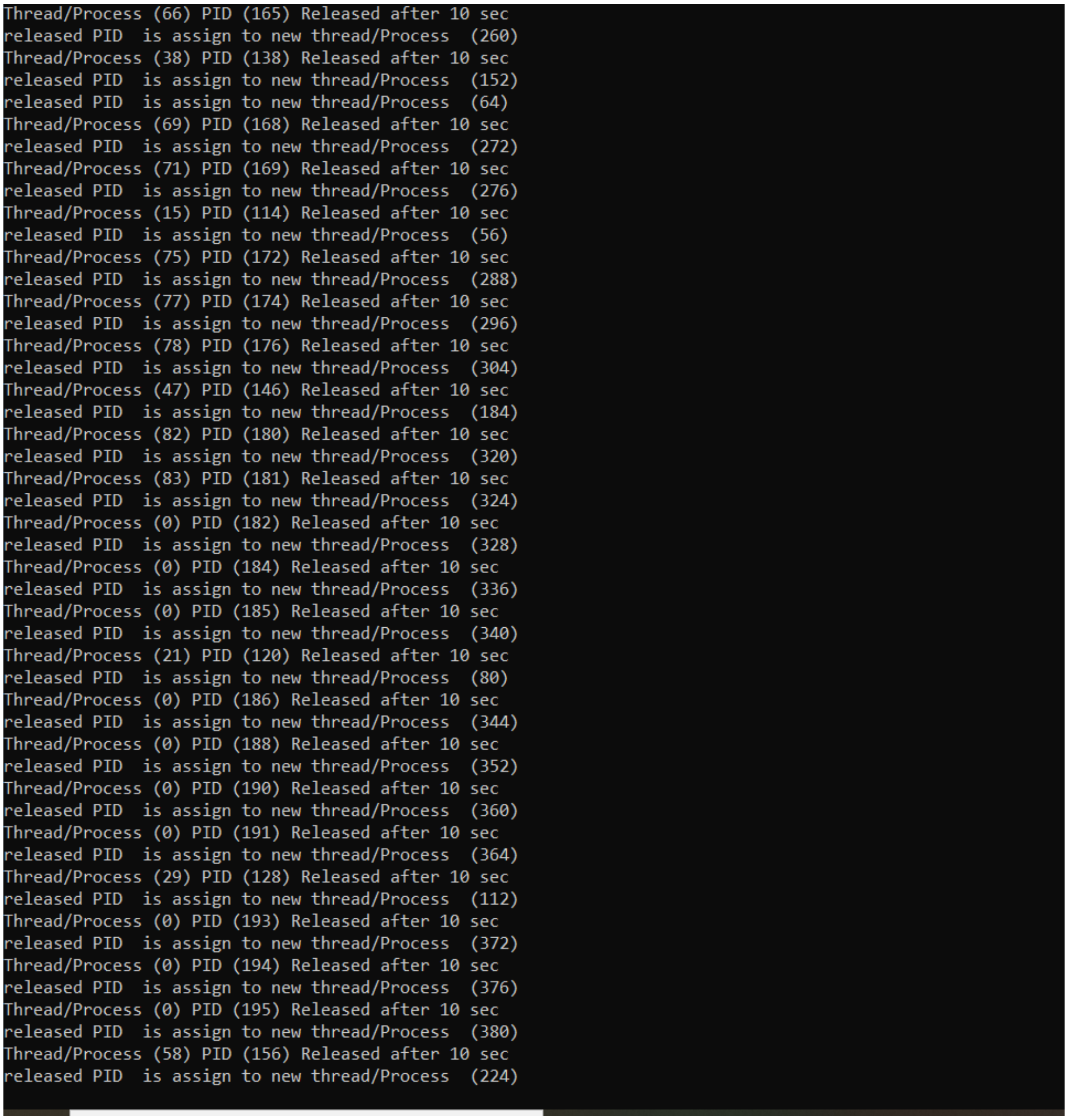
OUTPUT :- IT WILL DISPLAY 100 HREADS WITH HEIR CORRESPONDING PIDS WHICH ARE UNIQUE..



```
Thread/Process  52 is allocated with PID 152
Thread/Process  54 is allocated with PID 153
Thread/Process  55 is allocated with PID 154
Thread/Process  18 is allocated with PID 118
Thread/Process  0 is allocated with PID 197
Thread/Process  58 is allocated with PID 157
Thread/Process  59 is allocated with PID 158
Thread/Process  61 is allocated with PID 159
Thread/Process  62 is allocated with PID 160
Thread/Process  62 is allocated with PID 161
Thread/Process  62 is allocated with PID 162
Thread/Process  64 is allocated with PID 163
Thread/Process  38 is allocated with PID 137
Thread/Process  66 is allocated with PID 165
Thread/Process  38 is allocated with PID 138
Thread/Process  68 is allocated with PID 167
Thread/Process  69 is allocated with PID 168
Thread/Process  71 is allocated with PID 169
Thread/Process  72 is allocated with PID 170
Thread/Process  74 is allocated with PID 171
Thread/Process  75 is allocated with PID 172
Thread/Process  75 is allocated with PID 173
Thread/Process  77 is allocated with PID 174
Thread/Process  78 is allocated with PID 175
Thread/Process  78 is allocated with PID 176
Thread/Process  80 is allocated with PID 177
Thread/Process  47 is allocated with PID 146
Thread/Process  80 is allocated with PID 179
Thread/Process  82 is allocated with PID 180
Thread/Process  83 is allocated with PID 181
Thread/Process  0 is allocated with PID 182
Thread/Process  0 is allocated with PID 183
Thread/Process  0 is allocated with PID 184
Thread/Process  0 is allocated with PID 185
Thread/Process  0 is allocated with PID 186
Thread/Process  0 is allocated with PID 187
Thread/Process  0 is allocated with PID 188
Thread/Process  0 is allocated with PID 189
Thread/Process  0 is allocated with PID 190
Thread/Process  0 is allocated with PID 191
Thread/Process  0 is allocated with PID 192
Thread/Process  0 is allocated with PID 193
Thread/Process  0 is allocated with PID 194
Thread/Process  0 is allocated with PID 195
Thread/Process  0 is allocated with PID 196
Thread/Process  58 is allocated with PID 156
Thread/Process  0 is allocated with PID 198
Thread/Process  0 is allocated with PID 199
Thread/Process  0 is allocated with PID 200
```

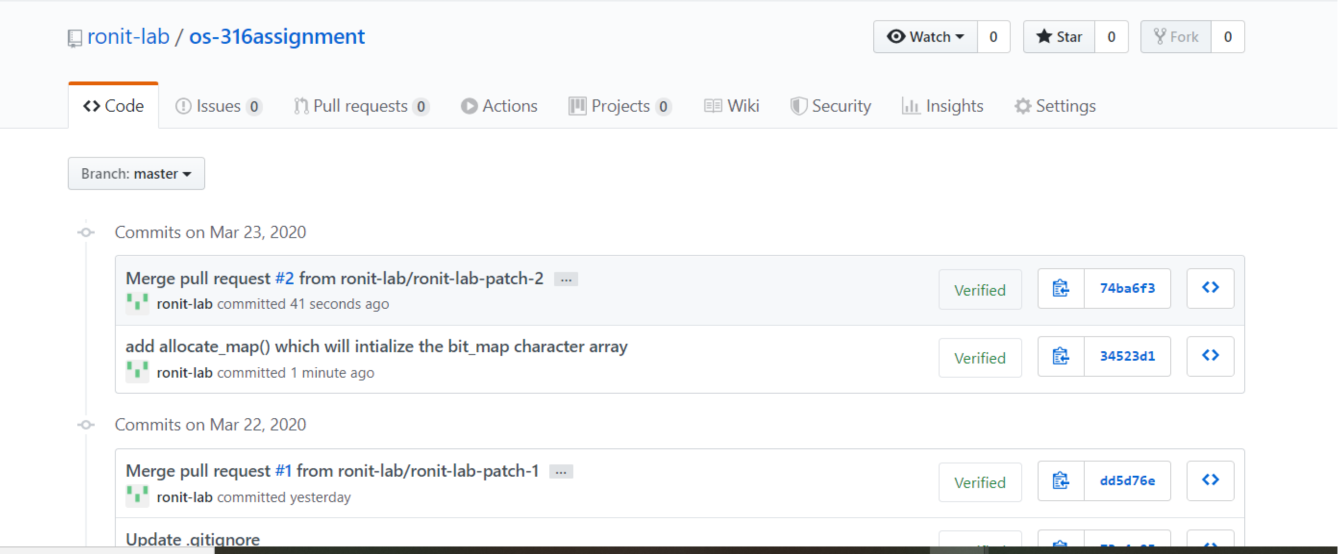
THREADS SLEEP FOR 10 SEC HAN PID ARE RELEASED BY HRADS AFTER 10 SEC …….

RELEASED PID ARE ASSIGNED TO NEW PROCESS.



```
Thread/Process (66) PID (165) Released after 10 sec
released PID  is assign to new thread/Process  (260)
Thread/Process (38) PID (138) Released after 10 sec
released PID  is assign to new thread/Process  (152)
released PID  is assign to new thread/Process  (64)
Thread/Process (69) PID (168) Released after 10 sec
released PID  is assign to new thread/Process  (272)
Thread/Process (71) PID (169) Released after 10 sec
released PID  is assign to new thread/Process  (276)
Thread/Process (15) PID (114) Released after 10 sec
released PID  is assign to new thread/Process  (56)
Thread/Process (75) PID (172) Released after 10 sec
released PID  is assign to new thread/Process  (288)
Thread/Process (77) PID (174) Released after 10 sec
released PID  is assign to new thread/Process  (296)
Thread/Process (78) PID (176) Released after 10 sec
released PID  is assign to new thread/Process  (304)
Thread/Process (47) PID (146) Released after 10 sec
released PID  is assign to new thread/Process  (184)
Thread/Process (82) PID (180) Released after 10 sec
released PID  is assign to new thread/Process  (320)
Thread/Process (83) PID (181) Released after 10 sec
released PID  is assign to new thread/Process  (324)
Thread/Process (0) PID (182) Released after 10 sec
released PID  is assign to new thread/Process  (328)
Thread/Process (0) PID (184) Released after 10 sec
released PID  is assign to new thread/Process  (336)
Thread/Process (0) PID (185) Released after 10 sec
released PID  is assign to new thread/Process  (340)
Thread/Process (21) PID (120) Released after 10 sec
released PID  is assign to new thread/Process  (80)
Thread/Process (0) PID (186) Released after 10 sec
released PID  is assign to new thread/Process  (344)
Thread/Process (0) PID (188) Released after 10 sec
released PID  is assign to new thread/Process  (352)
Thread/Process (0) PID (190) Released after 10 sec
released PID  is assign to new thread/Process  (360)
Thread/Process (0) PID (191) Released after 10 sec
released PID  is assign to new thread/Process  (364)
Thread/Process (29) PID (128) Released after 10 sec
released PID  is assign to new thread/Process  (112)
Thread/Process (0) PID (193) Released after 10 sec
released PID  is assign to new thread/Process  (372)
Thread/Process (0) PID (194) Released after 10 sec
released PID  is assign to new thread/Process  (376)
Thread/Process (0) PID (195) Released after 10 sec
released PID  is assign to new thread/Process  (380)
Thread/Process (58) PID (156) Released after 10 sec
released PID  is assign to new thread/Process  (224)
```

NO OF COMMITS OR REVISIONS IN CODE :-



Commits on Mar 23, 2020

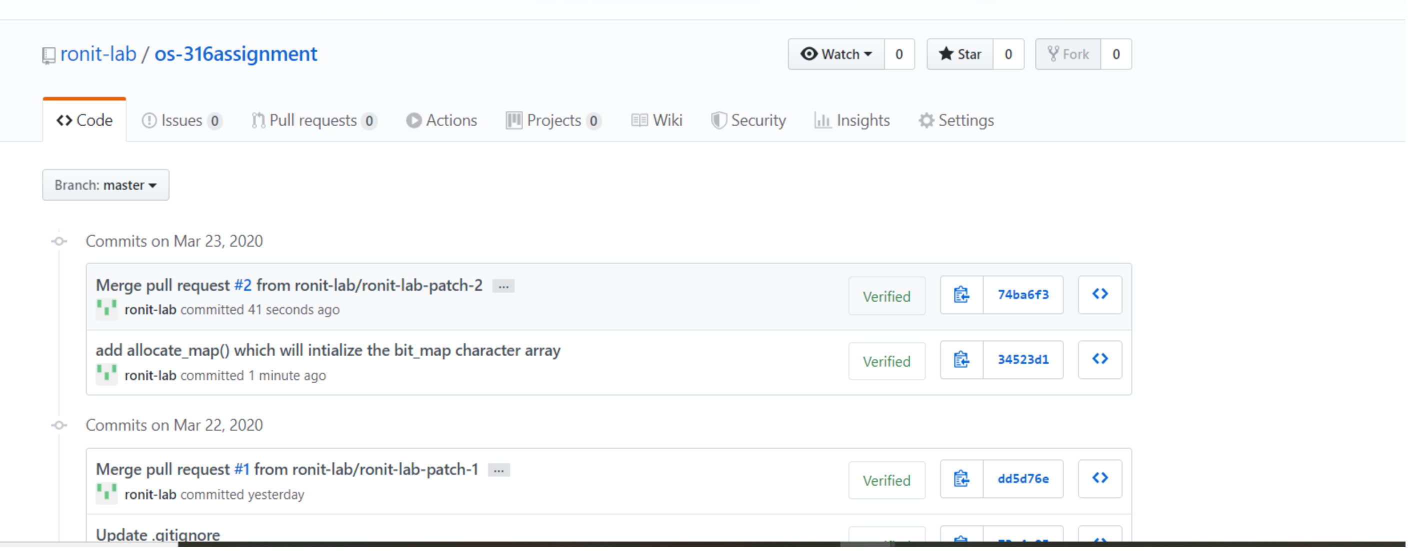
Merge pull request #2 from ronit-lab/ronit-lab-patch-2 …
ronit-lab committed 41 seconds ago — Verified — 74ba6f3

add allocate_map() which will intialize the bit_map character array
ronit-lab committed 1 minute ago — Verified — 34523d1

Commits on Mar 22, 2020

Merge pull request #1 from ronit-lab/ronit-lab-patch-1 …
ronit-lab committed yesterday — Verified — dd5d76e

Update .gitignore


ronit-lab / os-316assignment

Commits on Mar 23, 2020

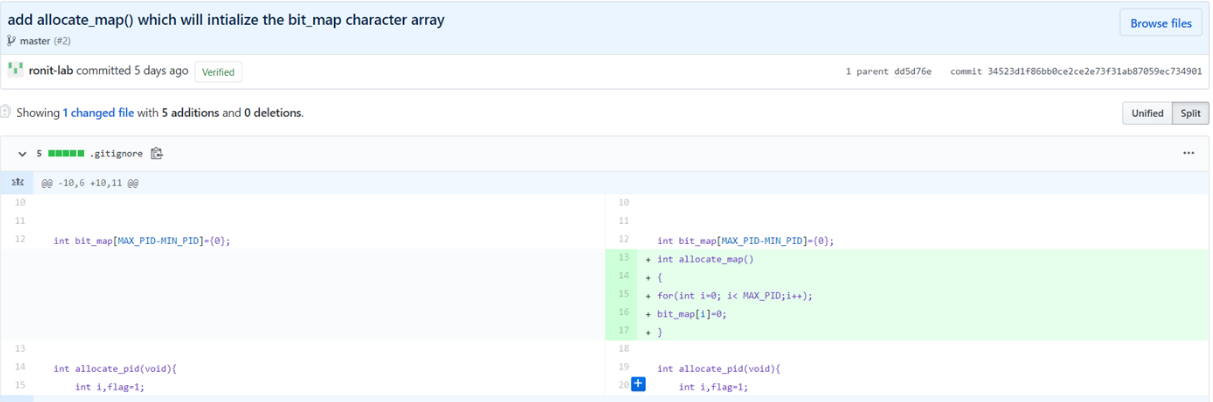
Merge pull request #2 from ronit-lab/ronit-lab-patch-2 …
ronit-lab committed 41 seconds ago — Verified — 74ba6f3

add allocate_map() which will intialize the bit_map character array
ronit-lab committed 1 minute ago — Verified — 34523d1

Commits on Mar 22, 2020

Merge pull request #1 from ronit-lab/ronit-lab-patch-1 …
ronit-lab committed yesterday — Verified — dd5d76e

Update .gitignore



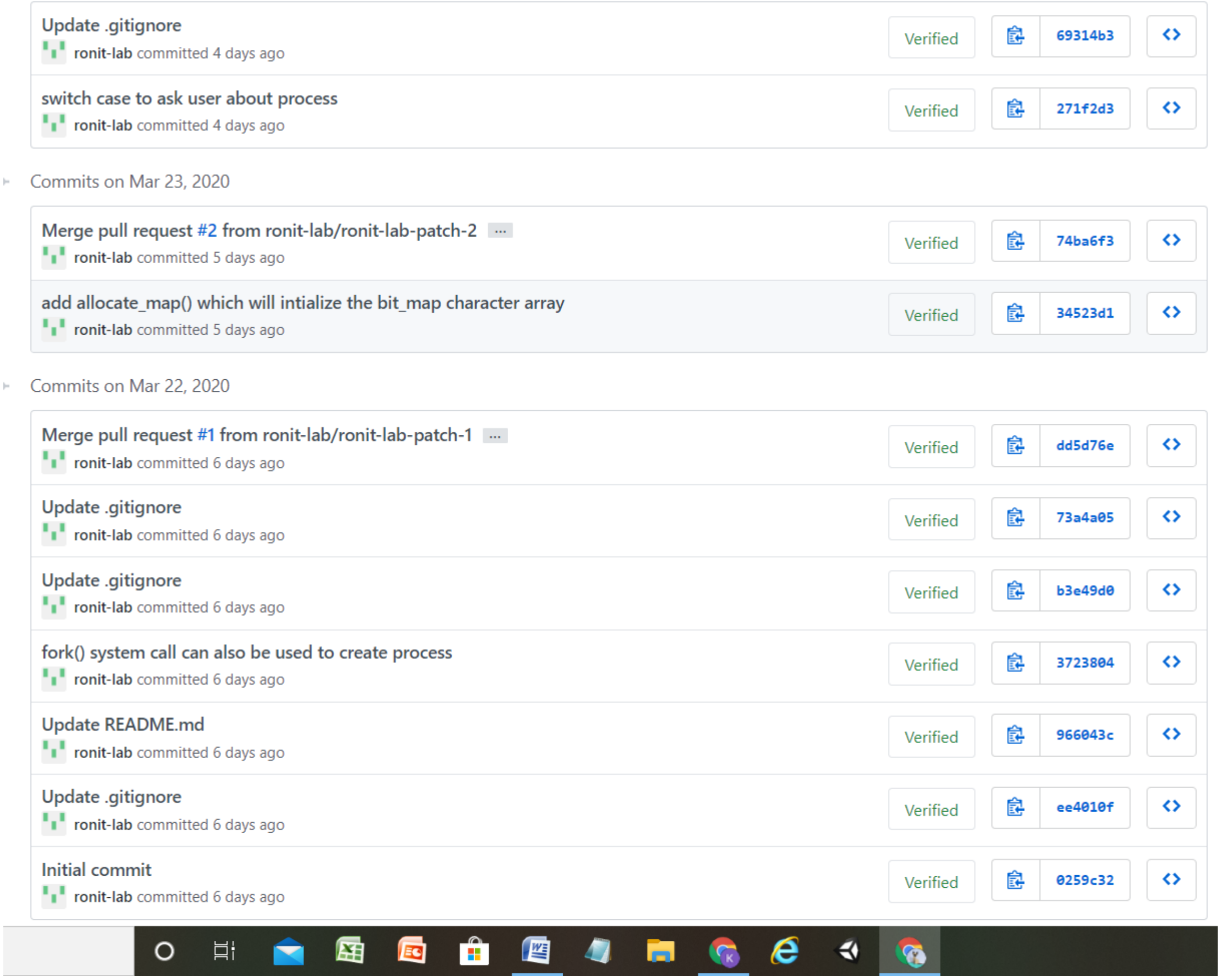
add allocate_map() which will intialize the bit_map character array
master (#2)

ronit-lab committed 5 days ago Verified   1 parent dd5d76e   commit 34523d1f86bb0ce2ce2e73f31ab87059ec734901

Showing 1 changed file with 5 additions and 0 deletions.

5 ■■■■■ .gitignore

```
@@ -10,6 +10,11 @@


int bit_map[MAX_PID-MIN_PID]={0};          int bit_map[MAX_PID-MIN_PID]={0};
                                           + int allocate_map()
                                           + {
                                           + for(int i=0; i< MAX_PID;i++);
                                           + bit_map[i]=0;
                                           + }

int allocate_pid(void){                    int allocate_pid(void){
    int i,flag=1;                              int i,flag=1;
```



Update .gitignore
ronit-lab committed 4 days ago — Verified 69314b3

switch case to ask user about process
ronit-lab committed 4 days ago — Verified 271f2d3

Commits on Mar 23, 2020

Merge pull request #2 from ronit-lab/ronit-lab-patch-2 …
ronit-lab committed 5 days ago — Verified 74ba6f3

add allocate_map() which will intialize the bit_map character array
ronit-lab committed 5 days ago — Verified 34523d1

Commits on Mar 22, 2020

Merge pull request #1 from ronit-lab/ronit-lab-patch-1 …
ronit-lab committed 6 days ago — Verified dd5d76e

Update .gitignore
ronit-lab committed 6 days ago — Verified 73a4a05

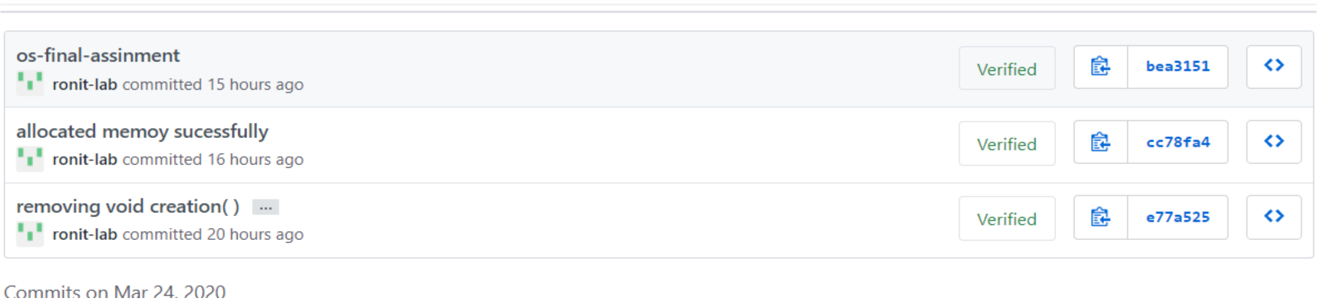
Update .gitignore
ronit-lab committed 6 days ago — Verified b3e49d0

fork() system call can also be used to create process
ronit-lab committed 6 days ago — Verified 3723804

Update README.md
ronit-lab committed 6 days ago — Verified 966043c

Update .gitignore
ronit-lab committed 6 days ago — Verified ee4010f

Initial commit
ronit-lab committed 6 days ago — Verified 0259c32



os-final-assinment
ronit-lab committed 15 hours ago — Verified bea3151

allocated memoy sucessfully
ronit-lab committed 16 hours ago — Verified cc78fa4

removing void creation( ) …
ronit-lab committed 20 hours ago — Verified e77a525

Commits on Mar 24, 2020

Commits link :-

https://github.com/ronit-lab/os-316assignment/commits/master