NAME: MOHAMMAD AMIR IDREESI

REG.NO: 11811970

EMAIL: amir23feb@gmail.com

GITHUB LINK: https://github.com/amir7587/amir7587

LOVELY PROFESSIONAL UNIVERSITY

Transforming Education Transforming India

PROBLEM: Q7. Create a Process Id (PID) manager that keeps track of free PIDs and ensures that no two active processes are having the same pid. Once a process terminates the PID manager may assigns its pid to new process. Use the following constants to identify the range of possible pid values: #define MIN PID 100 #define MAX PID 1000

You may use any data structure of your choice to represent the availability of process identifiers. One strategy is to adopt what Linux has done and use a bitmap in which a value of 0 at position i indicates that a process id of value i is available and a value of 1 indicates that the process id is currently in use. Implement the following API for obtaining and releasing a pid:

• int allocate map(void)—Creates and initializes a data structure for representing pid; 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 pid are in use) • 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, sleep for a random period of time, and then release the pid. (Sleeping for a random period of time approximates the typical pid usage in which a pid is assigned to a new process, the process executes and then terminates, and the pid is released on the process's termination.) .

CODE

```
#include<stdlib.h>
#include<time.h>
#include<unistd.h>
#include<pthread.h>
#include<sys/wait.h>
#include<stdio.h>

#define MIN_PID 100
#define MAX_PID 1000
#define NO_OF_P 100

int pid[MAX_PID-MIN_PID]={0};
int allocate_pid()
{
int i=0;
int state=1;
while(i<(MAX_PID-MIN_PID))
{
if(pid[i]==0)
{
pid[i]=1;
state=0;
break;
}
i++;
}
if(state==1)
{
return -1;
}
else
{
return i;
}
}
void release_pid(int id){
    pid[id]=0;
}
void *map(void *var){
    int tid =  *(( int* )var);

    int id = allocate_pid();



    if(id==-1){
        printf("SORRY NO PID IS CAN BE ALLOCATED  NOW");
    }



    else{
        printf("Thread [%3d] with  PID [%3d] \n",tid,id+MIN_PID);
        int stime=1+rand()%30;
        sleep(stime);
        printf("Thread [%3d] PID [%3d] Releasing this after  %d 
seconds\n",tid,id+MIN_PID,stime);
        release_pid(id);
    }
    pthread_exit(NULL);
}
```

```
int main(){

    pthread_t process[NO_OF_P];
    int i=0;

    srand(time(NULL));
    for(i=0; i<100; i++){
        if(pthread_create(&process[i],NULL,map,(void*)&i))
            return -1*printf("This thread Creation  %d cannot possible!!!\n",i);
    }

    for(i=0; i<100; i++)
        pthread_join(process[i],NULL);
    wait(NULL);
    return printf("\n****************COMPLETED***************\n");
}
```

1.Explain the problem in terms of operating system concept? (Max 200 word)

In the Given Problem the very basic keyword is Process, process is a program in execution.

Now Process Id: Process Id is a unique identifier assigned by the operating system.

According to the problem the process manager is responsible for assigning each process to the processor so that all the process is executed for a short period of time. When a process is being executed, it has full control over the processor.

When an application program is started by the user, the high-level scheduler of the OS loads all or some parts of the program code from the secondary storage to the memory. This helps create a PCB (process control block). It is created to hold process information like current status of the process and its location in the memory. Also, the OS maintains a process table separately in the memory that basically lists all the processes are loading or running currently. Therefore, when a new process is added, it is assigned a different PID (Process Identification Number) for better process management.

In this program multithreading is used so first of all the What is Thread?

A thread is a sequence of instructions within a program that can be executed independently of other code. It contains stack (local variables, functions, return values), a copy of registers and any thread specific data to allow them to schedule individually.

2.Write the algorithm for proposed solution of the assigned problem.

Algorithm: (Assuming Maximum and Minimum pids and no. of threads is already defined).

Repeat Steps 1 to 9 until process gets terminated.

Step 1: Create process and allocate process id to the process such that no two process is having the same pid.

Step 2: If process id is successfully allocated then return 1 else return -1 and exit with failure.

Step 3: Create multithread for the process.

Step 4: If process is not having the pids then thread creation will fail and exit with failure.

Step 5: Ensure that each thread will requests for the particular pids, releases the pid after tasks gets completed.

Step 6: When a thread is requesting for the pids it ensures that no other process is having the same id.

Step 7: A thread can go for sleep for random period of time so that it releases the pids.

Step 8: After thread can go for sleep another process is requesting for the pids can get the pids

Step 9: Release the pids after the process gets terminated.

Step 10: Exit

3.Calculate complexity of implemented algorithm. (Student must specify complexity of each line of code along with overall complexity)

```
int allocate_pid()
{
int i=0;
int state=1;
while(i<(MAX_PID-MIN_PID))  ⇐  (O(n))(where n is the diff between max and min
pids)
{
if(pid[i]==0)
{
pid[i]=1;
state=0;
break;
}
i++;
}
if(state==1)
{
return -1;
}
else
{
return i;
}
}
```

```
int main(){

    pthread_t process[NO_OF_P];
    int i=0;

    srand(time(NULL));
    for(i=0; i<100; i++){   //  ⟵  (O(n) where n is the no. of threads here the n is 100)
        if(pthread_create(&process[i],NULL,map,(void*)&i))
            return -1*printf("This thread Creation  %d cannot possible!!!\n",i);
    }

    for(i=0; i<100; i++)      ⟵     (O(n) where n is the no. of threads here the n is 100)
        pthread_join(process[i],NULL);
    //wait(NULL);
    return printf("\n****************COMPLETED***************\n");
}
```

Therefore the overall complexity of the code is (O(n+n+n))

i.e.: O(3n) which is O(n)

4.Explain all the constraints given in the problem. Attach the code snippet of the implemented constraint.

Constraint 1: Create a Process Id (PID) manager that keeps track of free PIDs and ensures that no two active processes are having the same pid.

```
Code: int allocate_pid()
{
int i=0;
int state=1; //here initialising the states of process id by ensuring that first the process is not having any pid
while(i<(MAX_PID-MIN_PID))
{
if(pid[i]==0)
{
pid[i]=1;
state=0;  // after allocation of pid changing its state to 0.
break;
}
i++;
}
if(state==1)
{
return -1;
}
else
{
return i;
}
}
```

Constraint 2: Once a process terminates the PID manager may assigns its pid to new process.

Sol: void release_pid(int id){

pid[id]=0; //here releasing the pid when the process gets terminated which indicates that process id is available for assignment to the other process.

}

Constraint 3: You will create a number of threads—for example, 100—and each thread will request a pid, sleep for a random period of time, and then release the pid.

Sol:
```
for(i=0; i<100; i++){
        if(pthread_create(&process[i],NULL,map,(void*)&i))
            return -1*printf("This thread Creation  %d cannot possible!!!\n",i);
    }


    for(i=0; i<100; i++)
        pthread_join(process[i],NULL);
    wait(NULL); // thread is waiting,


printf("Thread [%3d] with  PID [%3d] \n",tid,id+MIN_PID);
        int stime=1+rand()%30;
        sleep(stime); .// the thread will go to sleep for some time.
        printf("Thread [%3d] PID [%3d] Releasing this after  %d 
seconds\n",tid,id+MIN_PID,stime);
        release_pid(id); // here releasing the pid.
```

5.If you have implemented any additional algorithm to support the solution, explain the need and usage of the same.

Sol: No, I have implemented only with the one algorithm which I've mentioned at the starting.

6. Explain the boundary conditions of the implemented code.

Sol: Boundary Conditions:

Max Pid: 1000

Min Pid: 100

No_of_processes: 100

Process id must be in the range of 100-1000.

No two-process having same pid.

7.Explain all the test cases applied on the solution of assigned problem

1) Process id range must be in between 100-1000.

2) No two process is having the same pid,

3) If process finish execution it must deallocate the pid such that pid is assigned to some other processes.

4) The waiting and releasing time are less than 30.

8.Have you made minimum 5 revisions of solution on GitHub?

Sol: Yes, I've made the revisions of solutions by explaining some part of the code more precisely.

GitHub Link: https://github.com/amir7587/amir7587