

Process ID Manager

**─**

Nabin Kumar Adhikari

Roll No. : 51

Section : K1604

Registration Number : 11617707

Email : [nabinadhikari674@gmail.com](mailto:nabinadhikari674@gmail.com)

GitHub Link : <https://github.com/NabinAdhikari674/PID-Manager>

Code : Question Number 7

# 

# Overview

A Process ID manager which assigns process IDs to the threads requesting them.The IDs should be unique to each running threads and they are reusable when the thread releases the ID on it’s exit.The process IDs assigned are in a range from 100 to 1000.

It’s tested by the use of an multithreaded program.The programming in done in C and is compiled and tested in gcc compiler in the Ubuntu OS.

# Details And Explanation

Process Identifier (PID) is a unique number assigned to each process that is used by most of the operating system kernels to specifically identify an active process.Process IDs can also be used as parameters in order to manipulate the processes.These process IDs must be unique in order to precisely identify the process which otherwise may result in fatal errors in the system.

The problem given (Question Number 7) also deals with threads.Threads are the smallest unit or sequence of programmed instructions.Threads exist within a process and they can be managed independently by a [scheduler](https://en.wikipedia.org/wiki/Scheduling_(computing)) which is a part of an operating system.Each thread also has its own ID called as Thread Identifier (TID) which is used to identify threads within a process.In the project multithreading is also used.Multithreading is an ability of the [CPU](https://en.wikipedia.org/wiki/Central_processing_unit) to execute multiple threads concurrently or simultaneously.Multithreading allows multiple threads to exist within the context of a single process.These threads can share the resources provided to its process but are able to execute independently on their own.

# Problem Approach

At first allocateMap() function ensures that all the IDs are free for allocation within the range specified (100-1000).Once a thread is created it requests an PID from the allocatePID() function which in turn returns it with an process ID.The thread is put under sleep for random period of time.Whenever an thread is running its PID cannot be allocated to other threads.This is ensured by use of an binary array where used PIDs has its corresponding value 1 whereas the PIDs which are free have corresponding value of 0.At last when a thread exits its PID is released by releasePID() function and the corresponding PID can reused.

## 

# Algorithms

## allocatePID()

if(pid==NULL)

Print ”The PID Manager is Not Initialized”;

return -1;

Set pidNum=-1,x=0;

while(x==0)

pidNum=rand()%(MAX\_PID-MIN\_PID)+MIN\_PID;

if(map[pidNum]==0)

break;

map[pidNum]=1;

if(pidNum==-1)

Print ”Unable to allocate PID”;

return pidNum;

## 

## releasePID(int pidNum)

if(map[pidNum]==0)

Print ”The PID is already released or not In Use”;

return -1;

map[pidNum]=0;

Print “PID released”;

## 

## 

## allocateMap()

set i=0;

for(i=0;i<(MAX\_PID-MIN\_PID);i++)

pid[i]=0;

map[i]=0;

return 1;

## 

## \*thread()

int t;

t=allocatePID();

Print (’’Allocated PID is : %d“,t);

pthread\_yield(rand()%(20-5)+5); //sleep(rand()%3) //we could use sleep() but it makes   the execution time longer but [pthread\_yield()](http://man7.org/linux/man-pages/man3/pthread_yield.3.html) simulates   the same condition with better efficiency and less time

Print (”Releasing PID : %d”,t);

releasePID(t);

# 

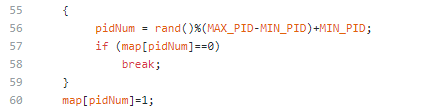
# Description With Code Snippets

The constraints used in the project are :

* MIN\_PID and MAX\_PID with values 100 and 1000 respectively.

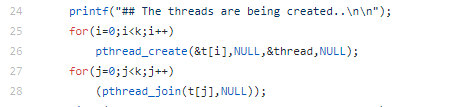


This restricts the PID Manager [allocatePID()] to allocate the PIDs within that range only.



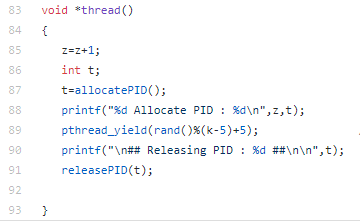
Here pidNum is returned as PIDs to process.The “map[pidNum]=1;” statement ensures that the corresponding value of allocated pid is 1 in Map array.

The Number of threads created can affect the performance of the program.If too many threads are created Memory Reference Error may occur.



Here ‘k’ is the number of threads to be created.

The threads are created as :



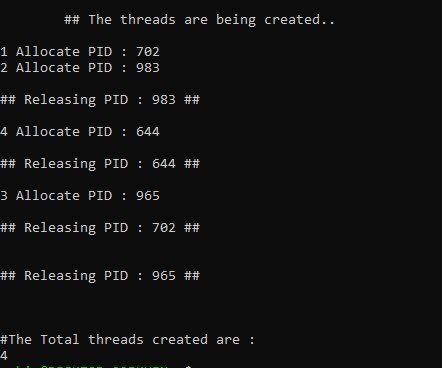
Here we could use sleep() instead of ”pthread\_yield(rand()%(k-5)+5);” but it makes the execution time longer but [pthread\_yield()](http://man7.org/linux/man-pages/man3/pthread_yield.3.html) simulates the same condition with better efficiency.

# Test Cases

The project must be compiled in gcc compiler with additional “-lpthread” or “-pthread” attribute in gcc.

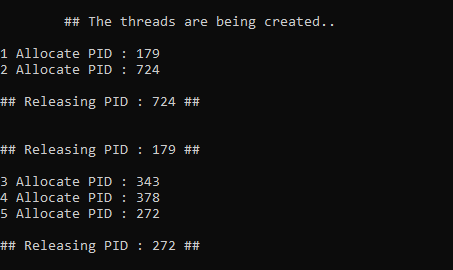
Among Many Test cases made following are some of them :

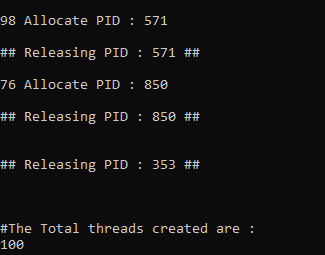
* When number of threads created was 4



Total 4 threads were created where threads are randomly slept and terminated after.As shown in the picture above the range of PID randomly generated is from 100 to 1000 and no two active threads have same PIDs.

* When number of threads created was 100





Total 100 threads were created but all of them cannot be shown here.All of them were allocated PIDs that no other active threads were allocated and that was in specified range.The PIDs were released when the threads are terminated.

# GitHub Repository

Link To Github : <https://github.com/NabinAdhikari674/PID-Manager>

The project was created on February 8th,2018.