5.40 Programming Exercise 3.20 required you to design a PID manager that

allocated a unique process identifier to each process. Exercise 4.20

required you to modify your solution to Exercise 3.20 by writing a

program that created a number of threads that requested and released

process identifiers. Now modify your solution to Exercise 4.20 by

ensuring that the data structure used to represent the availability of

process identifiers is safe from race conditions. Use Pthreads mutex

locks, described in Section 5.9.4.

#include <iostream>

#include <stdlib.h>

#include <pthread.h>

#include <windows.h>

using namespace std;

#define MIN\_PID 300

#define MAX\_PID 5000

int threadVar = 0;

pthread\_mutex\_t mutex;

struct pid\_tab

{

int pid;

bool bitmap;

}pidArr[4700];

int allocate\_map(void) //allocates bitmap values to the data structure

{

int i,j;

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

{

pidArr[j].pid = i;

pidArr[j].bitmap = 0;

}

if(i == MAX\_PID && j == 4700)

return 1;

else

return -1;

}

int allocate\_pid(void) //allocates a pid to the new process

{

for(int i = MIN\_PID, j =0; i <= MAX\_PID; i++, j++)

{

if(pidArr[j].bitmap == 0)

{

pidArr[j].pid = i;

pidArr[j].bitmap = 1;

return i;

break;

}

}

return -1;

}

void release\_pid(int pid) //releases pid

{

for(int i = 0; i <= 4700; i++)

{

if(pidArr[i].pid == pid)

{

pidArr[i].bitmap = 0;

}

}

}

/\* below function executes such that every thread only increments the threadVar by 1. Hence the output is numbers from 1 to 100 printed corresponding to each thread's execution.

The thread increments the value of threadVar by 1 and exits. Then the next thread increments by 1 again and exits. Every execution consists of a lock and unlock. \*/

void \* threadCall(void\* voidA) //function called by the created thread

{

int ret = allocate\_pid(); //allocates a pid

while (threadVar < 100)

{

pthread\_mutex\_lock(&mutex); //mutex lock occurs

if (threadVar >= 100)

{

pthread\_mutex\_unlock(&mutex);

break;

}

threadVar++; //threadVar increments at least once

Sleep(100);

cout<<"\n "<<threadVar;

//cout<<"\n "<<pidArr[threadVar].pid;

pthread\_mutex\_unlock(&mutex); //mutex now unlocked

}

Sleep(5);

release\_pid(ret); //pid released

}

int main()

{

int i =0;

pthread\_t thread[100];

cout<<"\n 100 threads created. Every thread will print the value of variable 'threadVar' and increment it by 1 with a delay of 100ms each process execution";

Sleep(3000); //delay only so that the above can be read in output screen before execution of the rest of the code

for(i = 0; i < 100; i++)

{

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

pthread\_create(&thread[i], NULL, threadCall, NULL);

threadCall(NULL);

}

for(i = 0; i < 100; i++)

{

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

pthread\_mutex\_destroy(&mutex);

}

return 0;

}