Name : Dev Adnani  
SID : 202212012

Subject : Systems Programming

Assignment : 10

Q1 : 202212012\_Lab10\_1.c

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

#include <math.h>

#include <pthread.h>

pthread\_mutex\_t myMutex;

int ptable[5][3];

int entry = 0;

int reRandomCheck(int processId);

void \*pTableWriteMutex(void \*str);

void \*pTableReadMutex(void \*str);

int main()

{

int ret;

pthread\_t threads[20];

ret = pthread\_mutex\_init(&myMutex, NULL);

for (int i = 0; i < 20; i++)

{

if (i % 2 == 0)

pthread\_create(&threads[i], NULL, pTableWriteMutex, NULL);

else

pthread\_create(&threads[i], NULL, pTableReadMutex, NULL);

}

for (int i = 0; i < 20; i++)

{

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

}

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

{

printf("%d %d %d\n", ptable[i][0], ptable[i][1], ptable[i][2]);

}

ret = pthread\_mutex\_destroy(&myMutex);

exit(0);

}

int reRandomCheck(int processId)

{

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

{

if (ptable[entry][0] == processId)

{

processId = rand() % 10;

return processId;

}

}

return processId;

}

void \*pTableWriteMutex(void \*str)

{

int processID = rand() % 10;

int pageNumber = (rand() % 1000) + 50;

int frequency = 1;

pthread\_mutex\_lock(&myMutex);

if (entry < 5)

{

processID = reRandomCheck(processID);

ptable[entry][0] = processID;

ptable[entry][1] = pageNumber;

ptable[entry][2] = frequency;

entry++;

}

else

{

int minFrequencyValue = ptable[0][2];

int minFrequencyPos = 0;

for (int i = 1; i < 5; i++)

{

if (ptable[i][2] < minFrequencyValue)

{

minFrequencyValue = ptable[i][2];

minFrequencyPos = i;

}

}

processID = reRandomCheck(processID);

ptable[minFrequencyPos][0] = processID;

ptable[minFrequencyPos][1] = pageNumber;

ptable[minFrequencyPos][2] = frequency;

}

pthread\_mutex\_unlock(&myMutex);

}

void \*pTableReadMutex(void \*str)

{

int processID = rand() % 10;

pthread\_mutex\_lock(&myMutex);

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

{

if (ptable[i][0] == processID)

{

ptable[i][2]++;

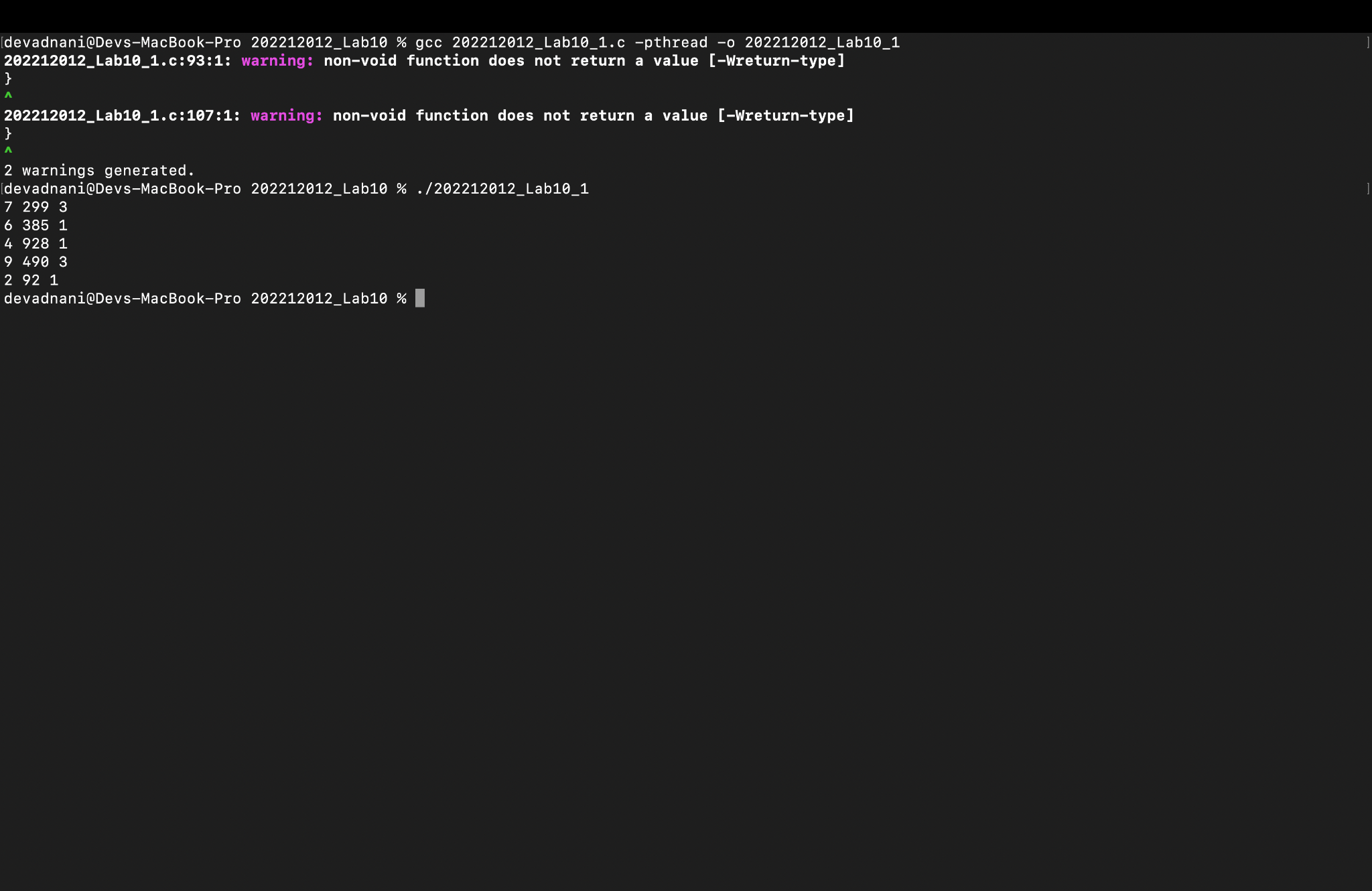
}

}

pthread\_mutex\_unlock(&myMutex);

}

Screenshot :



Q2 : 202212012\_Lab10\_2.c

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

#include <math.h>

#include <pthread.h>

pthread\_rwlock\_t rwLock;

int ptable[5][3];

int entry = 0;

int reRandomCheck(int processId);

void \*pTableWriteRW(void \*str);

void \*pTableReadRW(void \*str);

int main()

{

int ret;

pthread\_t threads[20];

ret = pthread\_rwlock\_init(&rwLock, NULL);

for (int i = 0; i < 20; i++)

{

if (i % 2 == 0)

pthread\_create(&threads[i], NULL, pTableWriteRW, NULL);

else

pthread\_create(&threads[i], NULL, pTableReadRW, NULL);

}

for (int i = 0; i < 20; i++)

{

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

}

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

{

printf("%d %d %d\n", ptable[i][0], ptable[i][1], ptable[i][2]);

}

ret = pthread\_rwlock\_destroy(&rwLock);

exit(0);

}

int reRandomCheck(int processId)

{

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

{

if (ptable[entry][0] == processId)

{

processId = rand() % 10;

return processId;

}

}

return processId;

}

void \*pTableWriteRW(void \*str)

{

int processID = rand() % 10;

int pageNumber = (rand() % 1000) + 50;

int frequency = 1;

pthread\_rwlock\_wrlock(&rwLock);

if (entry < 5)

{

processID = reRandomCheck(processID);

ptable[entry][0] = processID;

ptable[entry][1] = pageNumber;

ptable[entry][2] = frequency;

entry++;

}

else

{

int minFreqVal = ptable[0][2];

int minFreqPos = 0;

for (int i = 1; i < 5; i++)

{

if (ptable[i][2] < minFreqVal)

{

minFreqVal = ptable[i][2];

minFreqPos = i;

}

}

processID = reRandomCheck(processID);

ptable[minFreqPos][0] = processID;

ptable[minFreqPos][1] = pageNumber;

ptable[minFreqPos][2] = frequency;

}

pthread\_rwlock\_unlock(&rwLock);

}

void \*pTableReadRW(void \*str)

{

int processID = rand() % 10;

pthread\_rwlock\_rdlock(&rwLock);

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

{

if (ptable[i][0] == processID)

{

ptable[i][2]++;

}

}

pthread\_rwlock\_unlock(&rwLock);

}

Screenshot :



Q3: 202212012\_Lab10\_3.c

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

#include <math.h>

#include <pthread.h>

#include <semaphore.h>

sem\_t semaphore;

int p\_table[5][3];

int entry = 0;

int reRandomCheck(int processId);

void \*pTableWriteRW(void \*str);

void \*pTableReadRW(void \*str);

int main()

{

sem\_init(&semaphore, 0, 20);

pthread\_t threads[20];

for (int i = 0; i < 20; i++)

{

if (i % 2 == 0)

pthread\_create(&threads[i], NULL, pTableWriteRW, NULL);

else

pthread\_create(&threads[i], NULL, pTableReadRW, NULL);

}

for (int i = 0; i < 20; i++)

{

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

}

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

{

printf("%d %d %d\n", p\_table[i][0], p\_table[i][1], p\_table[i][2]);

}

exit(0);

}

int reRandomCheck(int processId)

{

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

{

if (p\_table[entry][0] == processId)

{

processId = rand() % 10;

return processId;

}

}

return processId;

}

void \*pTableWriteRW(void \*str)

{

int processID = rand() % 10;

int pageNumber = (rand() % 1000) + 50;

int frequency = 1;

sem\_wait(&semaphore);

if (entry < 5)

{

processID = reRandomCheck(processID);

p\_table[entry][0] = processID;

p\_table[entry][1] = pageNumber;

p\_table[entry][2] = frequency;

entry++;

}

else

{

int minFrequencyValue = p\_table[0][2];

int minFrequencyPos = 0;

for (int i = 1; i < 5; i++)

{

if (p\_table[i][2] < minFrequencyValue)

{

minFrequencyValue = p\_table[i][2];

minFrequencyPos = i;

}

}

processID = reRandomCheck(processID);

p\_table[minFrequencyPos][0] = processID;

p\_table[minFrequencyPos][1] = pageNumber;

p\_table[minFrequencyPos][2] = frequency;

}

sem\_post(&semaphore);

}

void \*pTableReadRW(void \*str)

{

int processID = rand() % 10;

sem\_wait(&semaphore);

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

{

if (p\_table[i][0] == processID)

{

p\_table[i][2]++;

}

}

sem\_post(&semaphore);

}

Screenshot :

