OS LAB # 8

M.Mubin Farid 21K-4827

Task 1)

#include <stdio.h>

#include <pthread.h>

#include <unistd.h>

void \*func(void \*arg){

char \*str="I am a string.";

pthread\_exit(str);

}

int main()

{

char \*str;

pthread\_t t;

pthread\_create(&t,NULL,func,NULL);

pthread\_join(t,(void\*\*)&str);

printf("%s\n",str);

}



Task 2)

#include <stdio.h>

#include <pthread.h>

#include <unistd.h>

int sum;

void \*func(void \*arg){

sum += \*((int\*)arg);

}

int main()

{

int arr[3]={1,3,5};

pthread\_t t1,t2,t3;

pthread\_create(&t1,NULL,func,(void\*)&arr[0]);

pthread\_create(&t2,NULL,func,(void\*)&arr[1]);

pthread\_create(&t3,NULL,func,(void\*)&arr[2]);

pthread\_join(t1,NULL);

pthread\_join(t2,NULL);

pthread\_join(t3,NULL);

printf("Sum: %d\n",sum);

}



Task 3)

#include <stdio.h>

#include <pthread.h>

#include <unistd.h>

int x=0;

pthread\_mutex\_t m;

void \*process1(void \*arg){

for(int i=0;i<1000000;i++){

x+=2;

}

}

void \*process2(void \*arg){

for(int i=0;i<1000000;i++){

x\*=1;

}

}

void \*process1mtx(void \*arg){

pthread\_mutex\_lock(&m);

for(int i=0;i<1000000;i++){

x+=2;

}

pthread\_mutex\_unlock(&m);

}

void \*process2mtx(void \*arg){

pthread\_mutex\_lock(&m);

for(int i=0;i<1000000;i++){

x\*=1;

}

pthread\_mutex\_unlock(&m);

}

int main()

{

pthread\_t t1,t2,t3,t4;

pthread\_create(&t1,NULL,process1,NULL);

pthread\_create(&t2,NULL,process2,NULL);

pthread\_join(t1,NULL);

pthread\_join(t2,NULL);

printf("Without Mutex: %d\n",x);

sleep(1);

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

x=0;

pthread\_create(&t3,NULL,process1mtx,NULL);

pthread\_create(&t4,NULL,process2mtx,NULL);

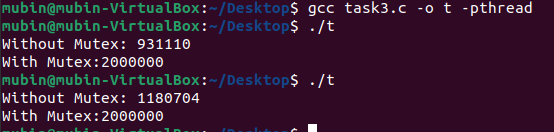
pthread\_join(t3,NULL);

pthread\_join(t4,NULL);

pthread\_mutex\_destroy(&m);

printf("With Mutex:%d\n",x);

}



When mutex locks were not applied we can see the difference in results as when both threads were trying to access the shared resource some instructions are wasted because if first thread reads variable value and second thread reads and applies operation on it then, first thread would still update the old value which it had already read, thus wasting second thread’s applied opertion.

The race condition is prevented with mutex locks as the as after one thread applys the mutex\_lock and enters it’s critical section the other thread will be in a waiting state to enter it’s critical region which it will enter only after the first thread applys mutex\_unlock, thus no instructions are wasted.