Programming Assignment #04

Due: 11/08/2021, by 11:59pm

Experience the race conditions and context switching.

```
/*=========================================================*/
/* race.c --- for playing with ECE437 */
/*=========================================================*/
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
struct {int balance[2];} Bank={{100,100}}; //global variable defined
void* MakeTransactions() { //routine for thread execution
 int i, j, tmp1, tmp2, rint; double dummy;
 for (i=0; i < 100; i++) {
 rint = (rand()%30)-15;
 if (((tmp1=Bank.balance[0])+rint)>=0 && ((tmp2=Bank.balance[1])-rint)>=0) {
 Bank.balance[0] = tmp1 + rint;
 for (j=0; j < rint*1000; j++) {dummy=2.345*8.765/1.234;} // spend time on purpose
 Bank.balance[1] = tmp2 - rint;
}
}
 return NULL;
}
int main(int argc, char **argv) {
int i; void* voidptr=NULL; pthread_t tid[2];
 srand(getpid());
 printf("Init balances A:%d + B:%d ==> %d!\n",
 Bank.balance[0],Bank.balance[1],Bank.balance[0]+Bank.balance[1]);
 for (i=0; i<2; i++) if (pthread_create(&tid[i],NULL,MakeTransactions, NULL)) {
 perror("Error in thread creating\n"); return(1); }
 for (i=0; i<2; i++) if (pthread_join(tid[i], (void*)&voidptr)) {
 perror("Error in thread joining\n"); return(1);}
 printf("Let's check the balances A:%d + B:%d ==> %d ?= 200\n",
 Bank.balance[0],Bank.balance[1],Bank.balance[0]+Bank.balance[1]);
 return 0;
}
```

Q1) (25%) Compile then run the above code for 20-40 times. Write a paragraph to explain the results.

Q2) (25%) Use thread library calls (mutex lock and unlock) to modify the code in Q1) to remove any potential race conditions. Show your modification of the code and explain the outcome with your modification

Q3) (25%) Rewrite your code in Q1) replacing threads by processes.

- Instead of creating two threads to call “MakeTransactions”, you will use fork() to create a child process. Both parent and child processes will call procedure “MakeTransactions”.

- Since two processes will not share a common address space, you will need to rewrite code to allocate “Bank” as a shared variable (by applying shared memory IPC, see Slide M02c).
- Other parts (i.e., set up initial values, print the initial values and balance, and print the ending values and balance) stay the same.

Show your implementation code in the written report, compile then run your new process-based code for 20-40 times. Write a paragraph to explain if the race condition still exists.

Q4) (25%) Use semaphore calls to modify your code in Q3 in order to remove any potential race conditions. Show your modification of the code and explain the outcome with your modification. (You can use either named or unnamed semaphore)

Note:
You do NOT need to submit your source code, only submit a written report which may contain some copy/paste your code segments (you may highlight/using a different font color to indicate your modified codes).