Bankers.c

```c
#include <stdio.h>
#include <pthread.h>
#include <stdlib.h>
#include <time.h>
#include <unistd.h>

#define NUM_TELLERS 3
#define NUM_CUSTOMERS 10

int account_balance = 1000;
pthread_mutex_t balance_lock;
void* serve_customer(void* teller_id) {
    int id = *((int*)teller_id);
    // lets assume serving 10 customers
    for (int i = 0; i < NUM_CUSTOMERS; i++) {
        int transaction_type = rand() % 2;
        int transaction_amount = rand() % 200 + 1;
        pthread_mutex_lock(&balance_lock);
        if (transaction_type == 0) {
            account_balance += transaction_amount;
            printf("Teller %d deposited $%d.
                New balance: $%d\n", id, transaction_amount, account_balance);
        } else {
            if (account_balance >= transaction_amount) {
                account_balance -= transaction_amount;
                printf("Teller %d withdrew $%d. New balance: $%d\n",
                    id, transaction_amount, account_balance);
            } else {
                printf("Teller %d tried to withdraw $%d,
                    but insufficient funds. Balance: $%d\n", id, transaction_amount, account_balance);
            }
        }
        pthread_mutex_unlock(&balance_lock);
        // Simulate time taken to serve a customer
        usleep((rand() % 1500 + 500) * 1000);
    }
    return NULL;
}
```

```c
int main() {
    srand(time(NULL));
    pthread_mutex_init(&balance_lock, NULL);
    pthread_t tellers[NUM_TELLERS];
    int teller_ids[NUM_TELLERS];
    for (int i = 0; i < NUM_TELLERS; i++) {
        teller_ids[i] = i + 1;
        if (pthread_create(&tellers[i], NULL, serve_customer, (void*)&teller_ids[i]) != 0) {
            perror("Failed to create thread");
            return 1;
        }
    }
    for (int i = 0; i < NUM_TELLERS; i++) {
        pthread_join(tellers[i], NULL);
    }
    printf("Final account balance: $%d\n", account_balance);
    pthread_mutex_destroy(&balance_lock);
    return 0;
}
```

```
student@VW:~$ touch bankers.c
student@VW:~$ gedit bankers.c
student@VW:~$ gcc bankers.c -o out
student@VW:~$ ./out
Teller 1 deposited $8. New balance: $1008
Teller 2 deposited $94. New balance: $1102
Teller 3 withdrew $111. New balance: $991
Teller 1 deposited $67. New balance: $1058
Teller 2 withdrew $47. New balance: $1011
Teller 3 withdrew $156. New balance: $855
Teller 1 deposited $159. New balance: $1014
Teller 2 withdrew $100. New balance: $914
Teller 1 withdrew $117. New balance: $797
Teller 3 withdrew $168. New balance: $629
Teller 3 deposited $46. New balance: $675
Teller 2 deposited $140. New balance: $815
Teller 1 deposited $20. New balance: $835
Teller 2 deposited $116. New balance: $951
Teller 3 deposited $25. New balance: $976
Teller 3 deposited $33. New balance: $1009
Teller 1 deposited $39. New balance: $1048
Teller 2 withdrew $118. New balance: $930
Teller 3 withdrew $149. New balance: $781
Teller 2 deposited $112. New balance: $893
Teller 1 withdrew $4. New balance: $889
Teller 3 deposited $47. New balance: $936
Teller 2 withdrew $151. New balance: $785
Teller 1 deposited $35. New balance: $820
Teller 2 withdrew $120. New balance: $700
Teller 1 deposited $178. New balance: $878
Teller 3 deposited $156. New balance: $1034
Teller 1 deposited $113. New balance: $1147
Teller 2 withdrew $68. New balance: $1079
Teller 3 withdrew $150. New balance: $929
Final account balance: $929
student@VW:~$
```

sortarr.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

#define SIZE 10

int arr[SIZE];
pthread_mutex_t mutex;
int compare(const void *a, const void *b) {
    return (*(int *)a - *(int *)b);
}
void merge(int *arr, int *left, int left_size, int *right, int right_size) {
    int i = 0, j = 0, k = 0;
    while (i < left_size && j < right_size) {
        if (left[i] <= right[j]) {
            arr[k++] = left[i++];
        } else {
            arr[k++] = right[j++];
        }
    }
    while (i < left_size) {
        arr[k++] = left[i++];
    }
    while (j < right_size) {
        arr[k++] = right[j++];
    }
}
```

```c
void *sort_first_half(void *arg) {
    int half_size = SIZE / 2;
    qsort(arr, half_size, sizeof(int), compare);
    return NULL;
}

void *sort_second_half(void *arg) {
    int half_size = SIZE / 2;
    qsort(arr + half_size, half_size, sizeof(int), compare);
    return NULL;
}
```

```c
int main() {
    srand(time(NULL));
    for (int i = 0; i < SIZE; i++) {
        arr[i] = rand() % 100;
    }
    printf("Unsorted array:\n");
    for (int i = 0; i < SIZE; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
    pthread_t thread1, thread2;
    pthread_create(&thread1, NULL, sort_first_half, NULL);
    pthread_create(&thread2, NULL, sort_second_half, NULL);
    pthread_join(thread1, NULL);
    pthread_join(thread2, NULL);

    int *left = arr;
    int *right = arr + SIZE / 2;
    int left_size = SIZE / 2;
    int right_size = SIZE - left_size;

    int *merged = (int *)malloc(sizeof(int) * SIZE);
    merge(merged, left, left_size, right, right_size);

    printf("Merged sorted array:\n");
    for (int i = 0; i < SIZE; i++) {
        printf("%d ", merged[i]);
    }
    printf("\n");

    free(merged);

    return 0;
}
```

```
student@VW:~$ touch sortarr.c
student@VW:~$ gedit sortarr.c
student@VW:~$ gcc sortarr.c -o out
student@VW:~$ ./out
Unsorted array:
75 42 17 27 39 38 81 41 55 23
Merged sorted array:
17 23 27 38 39 41 42 55 75 81
```