1. 개요

환경 : Visual Studio Code에 WSL을 설치한 우분투 환경에서 실시했습니다.

출금이 일어날 때 출금액보다 잔액이 더 적으면 충분한 입금액이 들어올 때까지 기다리게 만들어야 합니다. 이때 입금과 출금 간 동기화를 해주기 위해 'pthread_cond_wait()', 'pthread_cond_signal()', pthread_mutex_lock()/unlock()'을 적절히 사용해야 합니다.

2. 실행

```
pyeon@DESKTOP-DSS6GPS:~/testwsl$ gcc test3.c -lpthread && ./a.out
                        Wait for a deposit
Deposit 7                                       7
                        Withdraw 4              3
                        Wait for a deposit
Deposit 6                                       9
                        Withdraw 8              1
                        Wait for a deposit
Deposit 6                                       7
                        Withdraw 4              3
                        Wait for a deposit
Deposit 3                                       6
                        Wait for a deposit
Deposit 10                                      16
                        Withdraw 7              9
                        Withdraw 2              7
                        Withdraw 3              4
                        Wait for a deposit
Deposit 1                                       5
                        Wait for a deposit
Deposit 10                                      15
                        Withdraw 8              7
                        Withdraw 4              3
                        Wait for a deposit
Deposit 1                                       4
                        Wait for a deposit
Deposit 7                                       11
                        Withdraw 7              4
                        Withdraw 3              1
                        Wait for a deposit
Deposit 2                                       3
                        Wait for a deposit
```

잔액이 부족할 때 출금이 일어날 경우 "wait for a deposit"이 표시됩니다. 잔액이 충분할 때

출금이 일어날 경우 '잔액 - 출금액'이 현재 잔액에 반영되며 입금이 일어나면 '잔액 + 입금액'이 잔액에 반영됩니다.

3. 전체 코드 설명

(전역선언)

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

#define true 1

int balance = 0;

// 뮤텍스와 컨디션 변수
pthread_mutex_t lock;
pthread_cond_t newDeposit;

// 함수 프로토타입
void* DepositThread(void* arg);
void* WithdrawThread(void* arg);
void withdraw(int amount);
void deposit(int amount);
```

global varaible로 balance를 선언해 입출금이 일어날 때 이 변수에 값의 감가가 일어납니다. pthread 뮤텍스와 컨디션 변수를 선언하고 함수의 프로토타입들을 선언했습니다.

(main 함수)

```
int main() {
    // 뮤텍스와 조건변수 초기화
    pthread_mutex_init(&lock, NULL);
    pthread_cond_init(&newDeposit, NULL);

    // 스레드 생성
    pthread_t depositThread, withdrawThread;
    pthread_create(&depositThread, NULL, DepositThread, NULL);
    pthread_create(&withdrawThread, NULL, WithdrawThread, NULL);

    pthread_join(depositThread, NULL);
    pthread_join(withdrawThread, NULL);

    // 뮤텍스와 조건변수 제거
    pthread_mutex_destroy(&lock);
    pthread_cond_destroy(&newDeposit);

    return 0;
}
```

lock과 newDeposit을 초기화 하고 depositThread로 입금용 스레드, withdrawThread로 출금용 스레드를 생성합니다. 메인함수가 위 두 스레드가 종료될 때까지 기다리도록 join을 사용했습니다.

(Thread 함수)

```
void* DepositThread(void* arg) {
    while (true) {
        deposit((rand() % 10) + 1);
        sleep(1);
    }
    return NULL;
}

void* WithdrawThread(void* arg) {
    while (true) {
        withdraw((rand() % 10) + 1);
    }
    return NULL;
}
```

DepositThread에서는 생성한 랜덤값을 deposit()함수에 전달, 호출하는 행위를 반복합니다. WithdrawThread에서는 생성한 랜덤값을 withdraw()함수에 전달, 호출 행위를 반복합니다.

(withdraw(), deposit() 함수)

```
void withdraw(int amount) {     // 출금
    pthread_mutex_lock(&lock);
    while (balance < amount) {
        printf("\t\t\tWait for a deposit\n");
        pthread_cond_wait(&newDeposit, &lock);
    }
    balance -= amount;
    printf("\t\t\tWithdraw %d\t\t%d\n", amount, balance);
    pthread_mutex_unlock(&lock);
}

void deposit(int amount) {      // 입금
    pthread_mutex_lock(&lock);
    balance += amount;
    printf("Deposit %d\t\t\t\t\t%d\n", amount, balance);
    pthread_cond_signal(&newDeposit);
    pthread_mutex_unlock(&lock);
}
```

withdraw와 deposit 함수에서는 각각 출금과 입금이 발생할 때 lock을 걸고 임계 영역으로 진입합니다. 임계 영역을 빠져나오면 unlock을 하여 다른 스레드에서 lock을 사용할 수 있도록 해야 합니다. 잔액이 부족한데 withdraw 함수에서 출금이 일어날 경우 충분한 입금이 일어날 때까지 해당 스레드를 대기시켜야 합니다. 이때 withdraw의 pthread_cond_wait는 조건 변수를 사용하여 스레드를 대기 상태로 전환하고, 뮤텍스를 잠금 해제합니다. deposit 함수에서 newDeposit 조건변수에 signal을 줄 때까지 대기하게 됩니다. 조건 변수가 신호를 받으면 스레드는 뮤텍스를 다시 잠그고 대기에서 깨어납니다. 출금 후 unlock()을 실시합니다. deposit에서 unlock해주기 전에는 deposit이 lock을 획득한 상태이기에, cond wait가 깨어나도 cond wait가 lock을 걸 수는 없습니다. 이러한 동작과정은 온전한 동기화가 일어나도록 보장해줍니다.

deposit의 pthread_cond_signal은 조건 변수에 신호하여 `pthread_cond_wait`에서 대기 중인 스레드 하나를 깨우는 역할을 합니다.