The Given Program:

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

char *messages[3] = {NULL, NULL, NULL};

void *messenger(void *p) {
    long tid = (long)p;
    char tmpbuf[100];

    for(int i=0; i<10; i++)
    {
        /* Sending a message */
        long int dest = (tid + 1) % 3;
        sprintf(tmpbuf,"Hello from Thread %ld!", tid);
        char *msg = strdup(tmpbuf);
        messages[dest] = msg;
        printf("Thread %ld sent the message to Thread %ld\n",tid, dest);

        /* Receiving a message */
        printf("Thread %ld received the message '%s'\n",tid, messages[tid]);
        free(messages[tid]);
        messages[tid] = NULL;
    }
    return NULL;
}

void main()
{
    pthread_t thrID1, thrID2, thrID3;

    pthread_create(&thrID1, NULL, messenger, (void *)0);
    pthread_create(&thrID2, NULL, messenger, (void *)1);
    pthread_create(&thrID3, NULL, messenger, (void *)2);
    pthread_join(thrID1, NULL);
    pthread_join(thrID2, NULL);
    pthread_join(thrID3, NULL);
}
```

its output

```
Thread 1 sent the message to Thread 2
Thread 1 received the message '(null)'
Thread 1 sent the message to Thread 2
Thread 1 received the message '(null)'
Thread 1 sent the message to Thread 2
Thread 1 received the message '(null)'
Thread 1 sent the message to Thread 2
Thread 1 received the message '(null)'
Thread 1 sent the message to Thread 2
Thread 1 received the message '(null)'
Thread 1 sent the message to Thread 2
Thread 1 received the message '(null)'
Thread 1 sent the message to Thread 2
Thread 1 received the message '(null)'
Thread 1 sent the message to Thread 2
Thread 1 received the message '(null)'
Thread 1 sent the message to Thread 2
Thread 1 received the message '(null)'
Thread 1 sent the message to Thread 2
Thread 1 received the message '(null)'
Thread 0 sent the message to Thread 1
Thread 0 received the message '(null)'
Thread 0 sent the message to Thread 1
Thread 0 received the message '(null)'
Thread 0 sent the message to Thread 1
Thread 0 received the message '(null)'
Thread 0 sent the message to Thread 1
Thread 0 received the message '(null)'
Thread 0 sent the message to Thread 1
Thread 0 received the message '(null)'
Thread 0 sent the message to Thread 1
Thread 0 received the message '(null)'
Thread 0 sent the message to Thread 1
Thread 0 received the message '(null)'
Thread 0 sent the message to Thread 1
Thread 0 received the message '(null)'
Thread 0 sent the message to Thread 1
Thread 0 received the message '(null)'
Thread 0 sent the message to Thread 1
Thread 0 received the message '(null)'
Thread 2 sent the message to Thread 0
Thread 2 received the message 'Hello from Thread 1!'
Thread 2 sent the message to Thread 0
Thread 2 received the message '(null)'
Thread 2 sent the message to Thread 0
Thread 2 received the message '(null)'
Thread 2 sent the message to Thread 0
Thread 2 received the message '(null)'
Thread 2 sent the message to Thread 0
Thread 2 received the message '(null)'
Thread 2 sent the message to Thread 0
Thread 2 received the message '(null)'
Thread 2 sent the message to Thread 0
Thread 2 received the message '(null)'
Thread 2 sent the message to Thread 0
Thread 2 received the message '(null)'
Thread 2 sent the message to Thread 0
Thread 2 received the message '(null)'
Thread 2 sent the message to Thread 0
Thread 2 received the message '(null)'
```

This is because the `messages` is shared across all the threads.

Bg Wait

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

// Shared state
char *messages[3] = {NULL, NULL, NULL};

// The ID of the thread that is currently allowed to be the active thread (Receive THEN Send)
int turn = 0;

void *messenger(void *p) {
    long tid = (long)p;
    char tmpbuf[100];

    for(int i = 0; i < 10; i++) {
        long int dest = (tid + 1) % 3;

        while(turn != tid) { }

        while(messages[tid] == NULL) {}

        printf("Thread %ld RECEIVED the message '%s'\n", tid, messages[tid]);
        free(messages[tid]);
        messages[tid] = NULL;

        while(messages[dest] != NULL) {}

        sprintf(tmpbuf, "Hello from Thread %ld, Cycle %d!", tid, i + 1);
        char *msg = strdup(tmpbuf);
        messages[dest] = msg;

        printf("Thread %ld SENT the message to Thread %ld\n", tid, dest);

        turn = (tid + 1) % 3;
    }
    return NULL;
}

int main() {
    pthread_t thrID1, thrID2, thrID3;

    printf("Starting 10 cycles of strict communication (RECEIVE THEN SEND):\n");
    printf("Main sends to T0 to initiate the cycle.\n\n");

    // Create threads with IDs 0, 1, and 2
    pthread_create(&thrID1, NULL, messenger, (void *)0);
    pthread_create(&thrID2, NULL, messenger, (void *)1);
    pthread_create(&thrID3, NULL, messenger, (void *)2);

    // INITIATE THE CYCLE:
    // Send the first message to Thread 0's slot.
    // Thread 0 will be waiting for this message after it acquires the turn.
    char *msg = strdup("Initial message from main!");
    messages[0] = msg;

    // Wait for all threads to finish
    pthread_join(thrID1, NULL);
    pthread_join(thrID2, NULL);
    pthread_join(thrID3, NULL);

    printf("\nAll 10 cycles completed successfully.\n");
```

```
    return 0;
}
```

Mutexes

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

char *messages[3] = {NULL, NULL, NULL};

pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;

pthread_cond_t can_act[3] = {
    PTHREAD_COND_INITIALIZER,
    PTHREAD_COND_INITIALIZER,
    PTHREAD_COND_INITIALIZER
};

int current_turn = 0;
#define MAX_CYCLES 10

void *messenger(void *p) {
    long tid = (long)p;
    char tmpbuf[100];

    for(int i = 0; i < MAX_CYCLES; i++) {
        long int dest = (tid + 1) % 3;

        pthread_mutex_lock(&mutex);

        while (current_turn != tid || messages[tid] == NULL) {
            pthread_cond_wait(&can_act[tid], &mutex);
        }

        printf("Thread %ld RECEIVED the message '%s'\n", tid, messages[tid]);
        free(messages[tid]); messages[tid] = NULL;

        sprintf(tmpbuf, "Hello from Thread %ld, Cycle %d!", tid, i + 1);
        char *msg = strdup(tmpbuf);
        messages[dest] = msg;

        printf("Thread %ld SENT the message to Thread %ld\n", tid, dest);

        current_turn = dest;

        pthread_cond_signal(&can_act[dest]);

        pthread_mutex_unlock(&mutex);
    }

    return NULL;
}

int main()
{
    pthread_t thrID1, thrID2, thrID3;

    printf("Starting %d cycles of strict RECEIVE-THEN-SEND communication, initiated by main.
\n\n", MAX_CYCLES);

    pthread_create(&thrID1, NULL, messenger, (void *)0);
    pthread_create(&thrID2, NULL, messenger, (void *)1);
    pthread_create(&thrID3, NULL, messenger, (void *)2);

    pthread_mutex_lock(&mutex);
    char *initial_msg = strdup("Initial message from main!");
```

```
    messages[0] = initial_msg;

    printf("Main sent the initial message to Thread 0\n");

    pthread_cond_signal(&can_act[0]);
    pthread_mutex_unlock(&mutex);

    pthread_join(thrID1, NULL);
    pthread_join(thrID2, NULL);
    pthread_join(thrID3, NULL);

    pthread_mutex_destroy(&mutex);
    for(int i = 0; i < 3; i++) {
        pthread_cond_destroy(&can_act[i]);
    }

    printf("\nAll %d cycles completed successfully.\n", MAX_CYCLES);

    return 0;
}
```

Semaphores

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

char *messages[3] = {NULL, NULL, NULL};

sem_t empty[3];
sem_t full[3];

sem_t turn_send[3];

#define MAX_CYCLES 10

void *messenger(void *p) {
    long tid = (long)p;
    char tmpbuf[100];

    for(int i = 0; i < MAX_CYCLES; i++) {
        long int dest = (tid + 1) % 3;

        sem_wait(&turn_send[tid]);

        sem_wait(&full[tid]);

        printf("Thread %ld RECEIVED the message '%s'\n", tid, messages[tid]);
        free(messages[tid]);
        messages[tid] = NULL;

        sem_post(&empty[tid]);


        sem_wait(&empty[dest]);

        sprintf(tmpbuf, "Hello from Thread %ld, Cycle %d!", tid, i + 1);
        char *msg = strdup(tmpbuf);

        messages[dest] = msg;
        printf("Thread %ld SENT the message to Thread %ld\n", tid, dest);

        sem_post(&full[dest]);

        sem_post(&turn_send[dest]);
    }

    return NULL;
}

int main()
{
    pthread_t thrID1, thrID2, thrID3;

    printf("Starting %d cycles of strict RECEIVE-THEN-SEND communication (Semaphores).\n\n",
MAX_CYCLES);

    for(int i = 0; i < 3; i++)
    {
        sem_init(&empty[i], 0, 1);
        sem_init(&full[i], 0, 0);

        sem_init(&turn_send[i], 0, 0);
    }
```

```c
    pthread_create(&thrID1, NULL, messenger, (void *)0);
    pthread_create(&thrID2, NULL, messenger, (void *)1);
    pthread_create(&thrID3, NULL, messenger, (void *)2);

    char *initial_msg = strdup("Initial message from main!");

    messages[0] = initial_msg;
    printf("Main sent the initial message to Thread 0\n");

    sem_post(&full[0]);
    sem_post(&turn_send[0]);

    pthread_join(thrID1, NULL);
    pthread_join(thrID2, NULL);
    pthread_join(thrID3, NULL);

    for(int i = 0; i < 3; i++)
    {
        sem_destroy(&empty[i]);
        sem_destroy(&full[i]);
        sem_destroy(&turn_send[i]);
    }

    printf("\nAll %d cycles completed successfully.\n", MAX_CYCLES);

    return 0;
}
```