OS Assignment 05 Solution

1. Write a program in C that creates multiple threads to perform concurrent tasks. Implement a mechanism for communication and synchronization between these threads to ensure data consistency and proper execution order.

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

#define BUFFER_SIZE 10

int buffer[BUFFER_SIZE];
int buffer_filled = 0; // Indicates whether the buffer is filled with random
numbers

pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t cond = PTHREAD_COND_INITIALIZER;

// Function prototypes
void *generate_numbers(void *arg);
void *sort_numbers(void *arg);
void *calculate_sum_and_average(void *arg);

int main() {
  pthread_t thread1, thread2, thread3;

  // Create threads
  pthread_create(&thread1, NULL, generate_numbers, NULL);
  pthread_create(&thread2, NULL, sort_numbers, NULL);
  pthread_create(&thread3, NULL, calculate_sum_and_average, NULL);

  // Wait for all threads to finish
  pthread_join(thread1, NULL);
  pthread_join(thread2, NULL);
  pthread_join(thread3, NULL);

  return 0;
}

// Thread 1: Generates random numbers and fills the buffer
void *generate_numbers(void *arg) {
  pthread_mutex_lock(&mutex);
  printf("Thread 1: Generating random numbers...\n");
  for (int i = 0; i < BUFFER_SIZE; i++) {
    buffer[i] = rand() % 100; // Generate random numbers between 0 and 99
    printf("%d ", buffer[i]);
  }
  printf("\n");

  buffer_filled = 1; // Mark the buffer as filled
  pthread_cond_signal(&cond); // Signal Thread 2 to start sorting
  pthread_mutex_unlock(&mutex);

  return NULL;
}

// Thread 2: Sorts the numbers in the buffer
void *sort_numbers(void *arg) {
  pthread_mutex_lock(&mutex);
  while (!buffer_filled) {
    pthread_cond_wait(&cond, &mutex); // Wait for Thread 1 to fill the buffer
  }

  printf("Thread 2: Sorting numbers...\n");
  // Simple bubble sort
  for (int i = 0; i < BUFFER_SIZE - 1; i++) {
    for (int j = 0; j < BUFFER_SIZE - i - 1; j++) {
      if (buffer[j] > buffer[j + 1]) {
        int temp = buffer[j];
        buffer[j] = buffer[j + 1];
        buffer[j + 1] = temp;
      }
    }
  }

  for (int i = 0; i < BUFFER_SIZE; i++) {
    printf("%d ", buffer[i]);
  }
  printf("\n");

  buffer_filled = 2; // Mark the buffer as sorted

  pthread_cond_signal(&cond); // Signal Thread 3 to start calculations
  pthread_mutex_unlock(&mutex);

  return NULL;
}

// Thread 3: Calculates the sum and average of the numbers
void *calculate_sum_and_average(void *arg) {
  pthread_mutex_lock(&mutex);
  while (buffer_filled < 2) {
    pthread_cond_wait(&cond, &mutex); // Wait for Thread 2 to sort the buffer
  }

  printf("Thread 3: Calculating sum and average...\n");
  int sum = 0;
  for (int i = 0; i < BUFFER_SIZE; i++) {
    sum += buffer[i];
  }

  double average = (double)sum / BUFFER_SIZE;
  printf("Sum: %d, Average: %.2f\n", sum, average);

  pthread_mutex_unlock(&mutex);
  return NULL;
}
```

Explanation:

1. **Thread 1: Generate Numbers**
   - Fills the buffer with random numbers.
   - Signals Thread 2 to start sorting once the buffer is filled.
2. **Thread 2: Sort Numbers**
   - Waits for the signal from Thread 1.
   - Sorts the buffer using a simple bubble sort algorithm.
   - Signals Thread 3 to start calculations once sorting is complete.
3. **Thread 3: Calculate Sum and Average**
   - Waits for the signal from Thread 2.
   - Calculates the sum and average of the sorted numbers.
   - Prints the results.

Output Example:

```
Thread 1: Generating random numbers...
38 54 67 93 23 21 83 5 3 85
Thread 2: Sorting numbers...
3 5 21 23 38 54 67 83 85 93
Thread 3: Calculating sum and average...
Sum: 472, Average: 47.20
```

Synchronization Mechanism:

- **Mutex (pthread_mutex_t)**: Ensures that only one thread can access shared resources at a time.
- **Condition Variable (pthread_cond_t)**: Used to signal between threads when a certain condition is met, such as the buffer being filled or sorted.

This program demonstrates how to create multiple threads and coordinate their execution using synchronization mechanisms in a simple operating system context.

2. Create a C program that implements a parallel merge sort algorithm on an array of integers. The program will utilize multiple threads to sort the array concurrently, ensuring thread safety during the merge process.

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

```c
#include <string.h>
#define MAX_THREADS 16

typedef struct {
  int *array;
  int left;
  int right;
} SortParams;

pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;

void merge(int *array, int left, int mid, int right) {
  int i, j, k;
  int n1 = mid - left + 1;
  int n2 = right - mid;
  int *L = (int *)malloc(n1 * sizeof(int));
  int *R = (int *)malloc(n2 * sizeof(int));

  for (i = 0; i < n1; i++)
    L[i] = array[left + i];

  for (j = 0; j < n2; j++)
    R[j] = array[mid + 1 + j];

  i = 0;
  j = 0;
  k = left;

  while (i < n1 && j < n2) {
    if (L[i] <= R[j]) {
      array[k++] = L[i++];
    } else {
      array[k++] = R[j++];
    }
  }
  while (i < n1) {
    array[k++] = L[i++];
  }
  while (j < n2) {
    array[k++] = R[j++];
  }

  free(L);
  free(R);
}
void *merge_sort(void *params) {
  SortParams *p = (SortParams *)params;
  int left = p->left;
  int right = p->right;

  if (left < right) {
    int mid = left + (right - left) / 2;
    SortParams leftParams = {p->array, left, mid};
    SortParams rightParams = {p->array, mid + 1, right};
    pthread_t leftThread, rightThread;
    printf("Thread created to sort subarray [%d, %d]\n", left, right);
    pthread_create(&leftThread, NULL, merge_sort, &leftParams);
    pthread_create(&rightThread, NULL, merge_sort, &rightParams);
    pthread_join(leftThread, NULL);
    pthread_join(rightThread, NULL);
    pthread_mutex_lock(&mutex);
    merge(p->array, left, mid, right);
    pthread_mutex_unlock(&mutex);
    printf("Merging subarrays [%d, %d] and [%d, %d]\n", left, mid, mid
+ 1, right);
    printf("Sorted subarray [%d, %d]:\n", left, right);
    for (int i = left; i <= right; i++) {
      printf("%d ", p->array[i]);
    }
    printf("\n");
  }
  pthread_exit(NULL);
}

int main() {
  int n;
  printf("Enter the number of elements: ");
  if (scanf("%d", &n) != 1 || n <= 0) {
    fprintf(stderr, "Invalid number of elements.\n");
    return EXIT_FAILURE;
  }

  int *array = (int *)malloc(n * sizeof(int));

  if (array == NULL) {
    perror("Failed to allocate memory");
    return EXIT_FAILURE;
  }

  printf("Enter the elements:\n");
  for (int i = 0; i < n; i++) {
    if (scanf("%d", &array[i]) != 1) {
      fprintf(stderr, "Invalid input.\n");
      free(array);
      return EXIT_FAILURE;
    }
  }

  SortParams params = {array, 0, n - 1};
  pthread_t sortThread;
  printf("Thread created to sort the entire array [0, %d]\n", n - 1);
  pthread_create(&sortThread, NULL, merge_sort, &params);
  pthread_join(sortThread, NULL);
  printf("Sorted array:\n");
  for (int i = 0; i < n; i++) {
    printf("%d ", array[i]);
  }
  printf("\n");
  free(array);
  pthread_mutex_destroy(&mutex);
  return EXIT_SUCCESS;
}
```

Input:

```
Enter the number of elements: 8
Enter the elements:
38 27 43 3 9 82 10 55
```

Output:

```
Thread created to sort the entire array [0, 7]
Thread created to sort subarray [0, 3]
Thread created to sort subarray [0, 1]
Thread created to sort subarray [0, 0]
Thread created to sort subarray [1, 1]
Merging subarrays [0, 0] and [1, 1]
Sorted subarray [0, 1]:
27 38
Thread created to sort subarray [2, 3]
Thread created to sort subarray [2, 2]
Thread created to sort subarray [3, 3]
Merging subarrays [2, 2] and [3, 3]
Sorted subarray [2, 3]:
3 43
Merging subarrays [0, 1] and [2, 3]
Sorted subarray [0, 3]:
3 27 38 43
Thread created to sort subarray [4, 7]
Thread created to sort subarray [4, 5]
Thread created to sort subarray [4, 4]
Thread created to sort subarray [5, 5]
Merging subarrays [4, 4] and [5, 5]
Sorted subarray [4, 5]:
9 10
Thread created to sort subarray [6, 7]
Thread created to sort subarray [6, 6]
Thread created to sort subarray [7, 7]
Merging subarrays [6, 6] and [7, 7]
Sorted subarray [6, 7]:
55 82
Merging subarrays [4, 5] and [6, 7]
Sorted subarray [4, 7]:
9 10 55 82
Merging subarrays [0, 3] and [4, 7]
Sorted array:
3 9 10 27 38 43 55 82
```