main.cpp

```cpp
#include "queue.h"
#include "tokenize.cpp"
#include "fileProcess.cpp"
#include<iostream>
using namespace std;
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
#include <string.h>



void intro(char* prog_name);
void pro_con(int proCount, int conCount, FILE* files[], int
file_count);

int main(int argc, char* argv[]) {
    int   proCount, conCount;
    FILE* files[MAX_FILES];
    int   fileCount;
    if (argc != 3)
       intro(argv[0]);
    proCount = strtol(argv[1], NULL, 10);
    conCount = strtol(argv[2], NULL, 10);
    getFiles(files, &fileCount);
    printf("prod_count = %d, cons_count = %d, file_count = %d\n",
        proCount, conCount, fileCount);
    pro_con(proCount, conCount, files, fileCount);

    return 0;
}


void intro(char* prog_name) {

    fprintf(stderr, "Enter it in accordance with the following
format. Then enter multiple file names: %s <producer count>
<consumer count>\n",
        prog_name);
```

```c
    exit(0);
}





void pro_con(int proCount, int conCount, FILE* files[],
    int fileCount) {
    int thread_count = proCount + conCount;
    struct linkNode* front = NULL;
    struct linkNode* rear = NULL;
    int done_sending = 0;

#  pragma omp parallel num_threads(thread_count) default(none)
shared(fileCount, front, rear, files, proCount, conCount,
done_sending)
    {
        int my_rank = omp_get_thread_num(), k;
        if (my_rank < proCount) {
            for (k = my_rank; k < fileCount; k += proCount) {
                //for debug
                printf("Thread %d is about to read file %d\n",
my_rank, k);
                readFiles(files[k], &front, &rear, my_rank);
                //for debug
               // Print_queue(my_rank, queue_head);

            }
 #pragma omp atomic
            done_sending++;

        }
        else {
            struct linkNode* tmp_node;
            //Print_queue(my_rank, queue_head);
            while (done_sending< proCount) {
                tmp_node = Dequeue(&front, &rear, my_rank);
                if (tmp_node != NULL) {
                    Tokenize(tmp_node->data, my_rank);
                    free(tmp_node);
                }
            }
            if (front != NULL) {
                if (tmp_node != NULL) {
                    Tokenize(tmp_node->data, my_rank);
                    free(tmp_node);
                }

            }
        }
    }
}
```

queue.h:

```
#pragma once
void Enqueue(char* line, struct linkNode** front, struct linkNode**
rear);
struct linkNode* Dequeue(struct linkNode** front, struct linkNode**
rear, int my_rank);
void Print_queue(int my_rank, struct linkNode** rear);
```

queue.cpp:

```
#include "queue.h"
#include<iostream>
using namespace std;
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
#include <string.h>

struct linkNode {
    char* data;
    struct linkNode* next;
};


void Enqueue(char* line, struct linkNode** front, struct linkNode**
rear) {
    struct linkNode* tmp_node = NULL;
```

```c
    tmp_node = (linkNode*)malloc(sizeof(struct linkNode));
    tmp_node->data = line;
    tmp_node->next = NULL;

#   pragma omp critical
    {
        if (*rear == NULL) {
            *front = tmp_node;
            *rear = tmp_node;
        }
        else {
            (*rear)->next = tmp_node;
            *rear = tmp_node;
        }
    }
}



struct linkNode* Dequeue(struct linkNode** front, struct linkNode**
rear, int my_rank) {
    struct linkNode* tmp_node = NULL;
    if (*front == NULL)
        return NULL;

#   pragma omp critical
    {
        if (*front == *rear && *front != NULL)
            *rear = (*rear)->next;

        if (*front != NULL) {
            tmp_node = *front;
            *front = (*front)->next;
        }
    }

    return tmp_node;
}

void Print_queue(int my_rank, struct linkNode* front) {
    struct linkNode* curr_p = front;
    printf("Thread %d queue = \n", my_rank);
#   pragma omp critical
    while (curr_p != NULL) {
        printf("%s", curr_p->data);
        curr_p = curr_p->next;
    }
    printf("\n");
}
```

tokenize.h:

```
#pragma once
void Tokenize(char* data, int my_rank);
char* strtok_r1(char* str, const char* delim, char** saveptr);
```

tokenize.cpp:

```
#include "tokenize.h"
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
#include <string.h>
void Tokenize(char* data, int my_rank) {
    char* my_token, * word;
    my_token = strtok_r1(data, " \t\n", &word);
    while (my_token != NULL) {
        printf("(consumer)Thread %d for word segmentation: %s\n",
my_rank, my_token);
        my_token = strtok_r1(NULL, " \t\n", &word);
    }
}



/*From GNU C library*/
char* strtok_r1(char* s, const char* delim, char** save_ptr) {
    char* token;
    if (s == NULL) s = *save_ptr;

    /* Scan leading delimiters.  */
    s += strspn(s, delim);
    if (*s == '\0')
```

```
        return NULL;

    /* Find the end of the token.  */
    token = s;
    s = strpbrk(token, delim);
    if (s == NULL)
        /* This token finishes the string.  */
        *save_ptr = strchr(token, '\0');
    else {
        /* Terminate the token and make *SAVE_PTR point past it.
*/
        *s = '\0';
        *save_ptr = s + 1;
    }
    return token;
}
```

fileProcess.h:

```
#pragma once
#include<iostream>
using namespace std;
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
#include <string.h>
const int MAX_FILES = 50;
const int MAX_CHAR = 1000;

void getFiles(FILE* files[], int* file_number);
void ReadFiles(FILE* file, struct linkNode** queue_head,struct
linkNode** queue_tail, int my_rank);
```

fileProcess.cpp

```
#include "fileProcess.h"
#include "queue.cpp"

void getFiles(FILE* files[], int* file_number) {
    int i = 0;
    char filename[MAX_CHAR];
    while (scanf("%s", filename) != -1) {
        files[i] = fopen(filename, "r");
        //for debug
        printf("file %d = %s\n", i, filename);
        if (files[i] == NULL) {
            fprintf(stderr, "Can't open file %s\n", filename);
            fprintf(stderr, "Please try again \n");
            exit(-1);
        }
        i++;
    }
    *file_number = i;
}


void readFiles(FILE* file, struct linkNode** queue_head,struct
linkNode** queue_tail, int my_rank) {
    char* line = (char*)malloc(MAX_CHAR * sizeof(char));
    while (fgets(line, MAX_CHAR, file) != NULL) {
        printf("Thread %d reads line: %s\n", my_rank, line);
        Enqueue(line, queue_head, queue_tail);
        line = (char*)malloc(MAX_CHAR * sizeof(char));
    }
    fclose(file);
}
```