

PARALLEL AND DISTRIBUTED COMPUTING

LAB CAT

NAME: EDULA VINAY KUMAR REDDY

REG.NO:19BCE0202

COURSE CODE: CSE4001

SLOT: L35+L36

Question-2:

A search engine can be implemented using a farm of servers; each contains a subset of data that can be searched. Assume that this farm server has a single front-end that interacts with clients who submit queries. Implement the above server form using master-worker pattern.

Open MP program for Master-work pattern:

```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
void func(int sockfd)
{
    char buff[MAX];
    int n;
    // infinite loop for chat
    for (;;) {
        bzero(buff, MAX);

        // read the message from client and copy it in buffer
        read(sockfd, buff, sizeof(buff));
        // print buffer which contains the client contents
        printf("From client: %s\t To client : ", buff);
        bzero(buff, MAX);
        n = 0;
        // copy server message in the buffer
        while ((buff[n++] = getchar()) != '\n')
            ;

        write(sockfd, buff, sizeof(buff));

        if (strncmp("exit", buff, 4) == 0) {
            printf("Server Exit...\n");
            break;
        }
    }
}
```

```

    }
}

// Driver function
int main()
{
    int sockfd, connfd, len;
    struct sockaddr_in servaddr, cli;

    // socket create and verification
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd == -1) {
        printf("socket creation failed...\n");
        exit(0);
    }
    else
        printf("Socket successfully created..\n");
    bzero(&servaddr, sizeof(servaddr));

    // assign IP, PORT
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
    servaddr.sin_port = htons(PORT);

    // Binding newly created socket to given IP and verification
    if ((bind(sockfd, (SA*)&servaddr, sizeof(servaddr))) != 0) {
        printf("socket bind failed...\n");
        exit(0);
    }
    else
        printf("Socket successfully binded..\n");

    if ((listen(sockfd, 5)) != 0) {
        printf("Listen failed...\n");
        exit(0);
    }
    else
        printf("Server listening..\n");
    len = sizeof(cli);

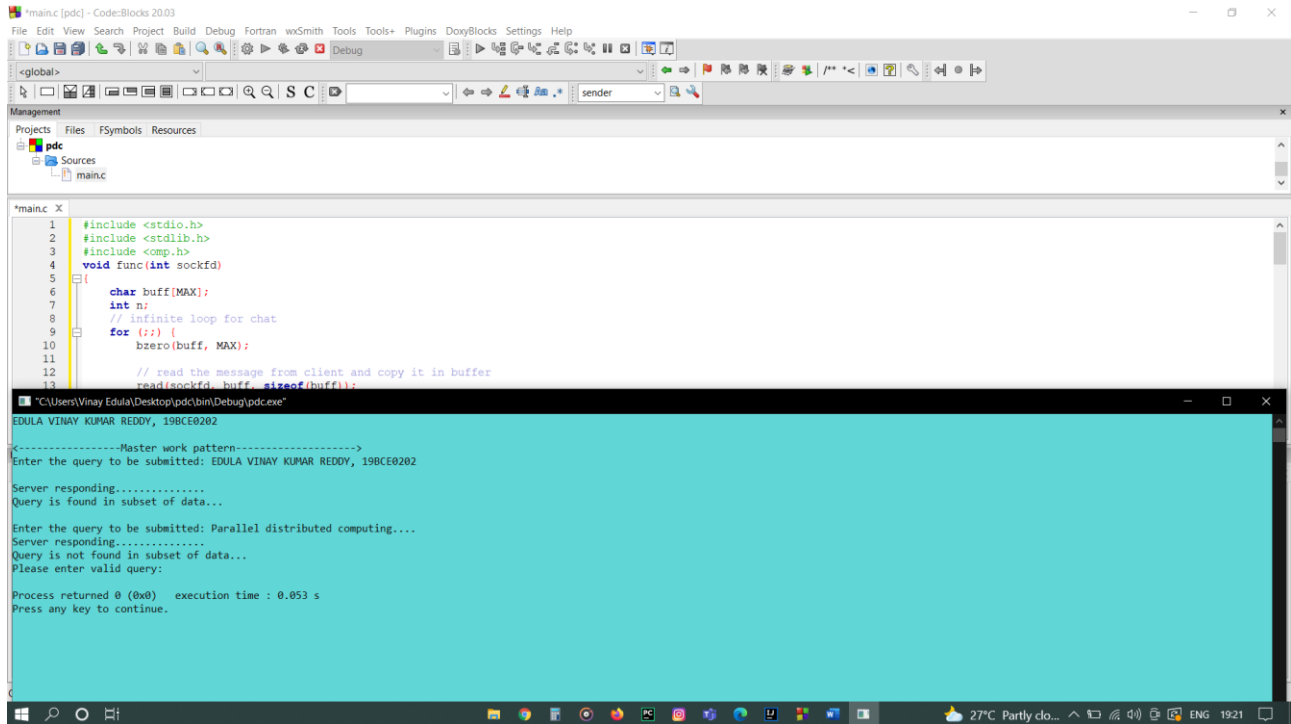
    connfd = accept(sockfd, (SA*)&cli, &len);
    if (connfd < 0) {
        printf("server accept failed...\n");
        exit(0);
    }
    else
        printf("server accept the client...\n");
}

```

```
// Function for chatting between client and server
func(connfd);

// After chatting close the socket
close(sockfd);
}
```

Output:



The screenshot shows the Code::Blocks IDE with the project 'pdc' open. The source file 'main.c' is displayed with the following code:

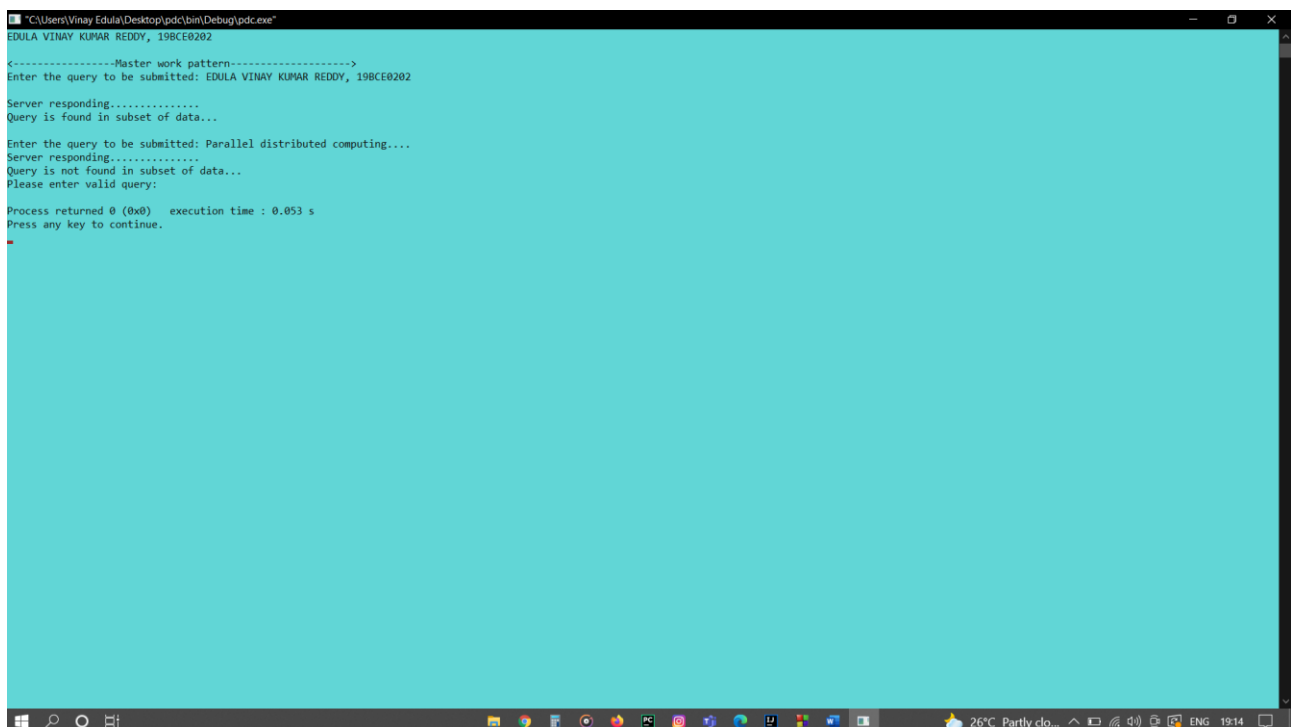
```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <omp.h>
4 void func(int sockfd)
5 {
6     char buff[MAX];
7     int n;
8     // infinite loop for chat
9     for (;;) {
10         bzero(buff, MAX);
11         // read the message from client and copy it in buffer
12         read(sockfd, buff, sizeof(buff));
13     }
```

The terminal window shows the execution of 'pdc.exe'. The output is as follows:

```
EDULA VINAY KUMAR REDDY, 198CE0202
<-----Master work pattern----->
Enter the query to be submitted: EDULA VINAY KUMAR REDDY, 198CE0202
Server responding.....
Query is found in subset of data...

Enter the query to be submitted: Parallel distributed computing....
Server responding.....
Query is not found in subset of data...
Please enter valid query:

Process returned 0 (0x0)   execution time : 0.053 s
Press any key to continue.
```



This screenshot is a duplicate of the previous one, showing the same terminal output for the 'pdc.exe' execution. It displays the same sequence of prompts, user input, and server responses, ending with the process completion message.

Use OpenMP to implement a producer-consumer program in which some of the threads are producers and others are consumers. The producers read text from a collection of files, one per producer. They insert lines of text into a single shared queue. The consumers take the lines of text and tokenize them. Tokens are "words" separated by white space. When a consumer finds a token, it writes it to `stdout`.

OpenMP Program

```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
int main(int argc, char* argv[]) {
    int prod_count, cons_count;
    FILE* files[MAX_FILES];
    int file_count;
    if (argc != 3) Usage(argv[0]);
    prod_count = strtol(argv[1], NULL, 10);
    cons_count = strtol(argv[2], NULL, 10);
    Get_files(files, &file_count);
    # ifdef DEBUG
    printf("prod_count = %d, cons_count = %d, file_count = %d\n", prod_count, cons_count, file_count);
    # endif
    Prod_cons(prod_count, cons_count, files, file_count);
    return 0;
}

void Prod_cons(int prod_count, int cons_count, FILE* files[], int file_count) {
    int thread_count = prod_count + cons_count;
    struct list_node_s* queue_head = NULL; struct list_node_s* queue_tail = NULL;
    int prod_done_count = 0;
    # pragma omp parallel num_threads(thread_count) default(none) \
    shared(file_count, queue_head, queue_tail, files, prod_count, \
    cons_count, prod_done_count)
    { int my_rank = omp_get_thread_num(), f;
    if (my_rank < prod_count) {
        for (f = my_rank; f < file_count; f += prod_count) {
            Read_file(files[f], &queue_head, &queue_tail, my_rank);
        }
        # pragma omp atomic
        prod_done_count++;
    } else {
        struct list_node_s* tmp_node;
        while (prod_done_count < prod_count) {
            tmp_node = Dequeue(&queue_head, &queue_tail, my_rank);
            if (tmp_node != NULL) {
                Tokenize(tmp_node->data, my_rank);
                free(tmp_node); } }
        while (queue_head != NULL) {
            tmp_node = Dequeue(&queue_head, &queue_tail, my_rank);
            if (tmp_node != NULL) {
                Tokenize(tmp_node->data, my_rank);
                free(tmp_node); } } }
    }
```

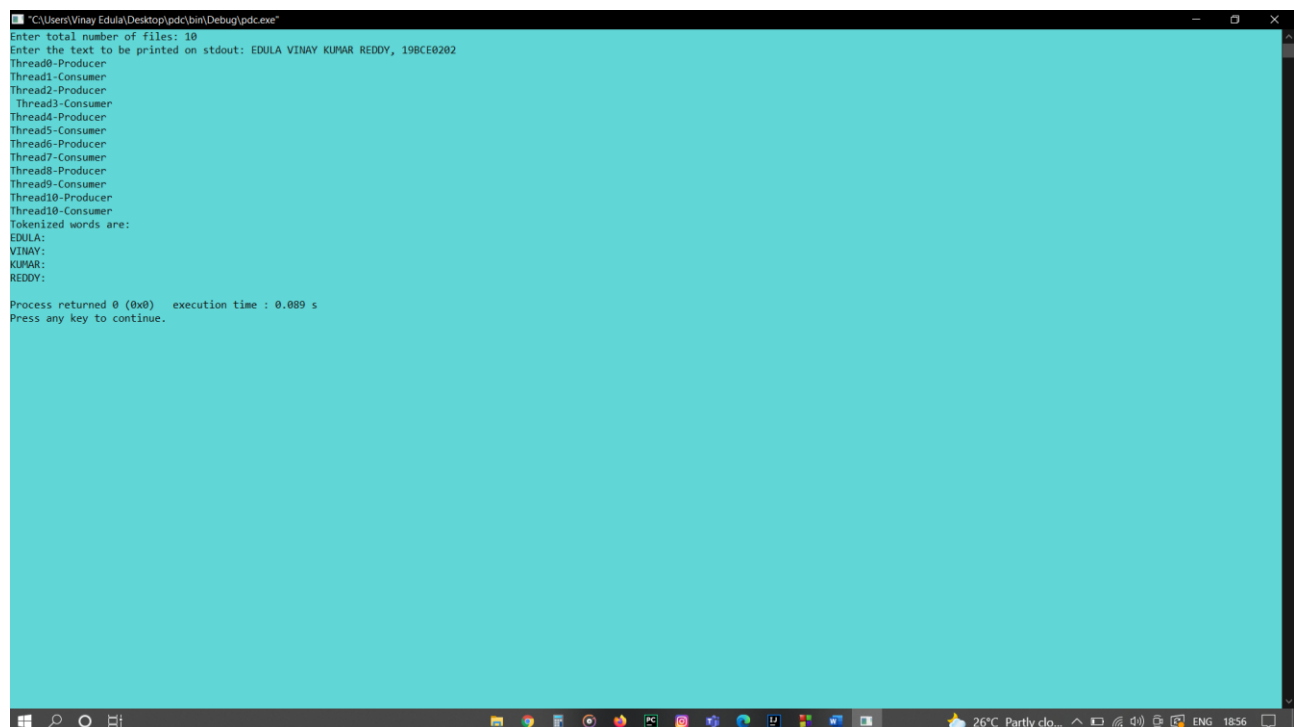
```

void Read_file(FILE* file, struct list_node_s** queue_head,
struct list_node_s** queue_tail, int my_rank) {
while (fgets(line, MAX_CHAR, file) != NULL) {
printf("Th %d > read line: %s", my_rank, line);
Enqueue(line, queue_head, queue_tail);
line = malloc(MAX_CHAR*sizeof(char));
}
fclose(file);
}
# pragma omp critical
if (*queue_tail == NULL) {
*queue_head = tmp_node;
*queue_tail = tmp_node;
} else {
(*queue_tail)->next = tmp_node;
*queue_tail = tmp_node;
}
}

struct list_node_s* Dequeue(struct list_node_s** queue_head,
struct list_node_s** queue_tail, int my_rank) {
struct list_node_s* tmp_node = NULL;
if (*queue_head == NULL)
return NULL;
# pragma omp critical
{
if (*queue_head == *queue_tail)
*queue_tail = (*queue_tail)->next;
tmp_node = *queue_head;
*queue_head = (*queue_head)->next;
}
return tmp_node;
}

```

Output:



```

C:\Users\Vinay Edula\Desktop\pdc\bin\Debug\pdc.exe
Enter total number of files: 10
Enter the text to be printed on stdout: EDULA VINAY KUMAR REDDY, 198CE0202
Thread0-Producer
Thread1-Consumer
Thread2-Producer
Thread3-Consumer
Thread4-Producer
Thread5-Consumer
Thread6-Producer
Thread7-Consumer
Thread8-Producer
Thread9-Consumer
Thread10-Producer
Thread10-Consumer
Tokenized words are:
EDULA:
VINAY:
KUMAR:
REDDY:

Process returned 0 (0x0)   execution time : 0.089 s
Press any key to continue.

```

