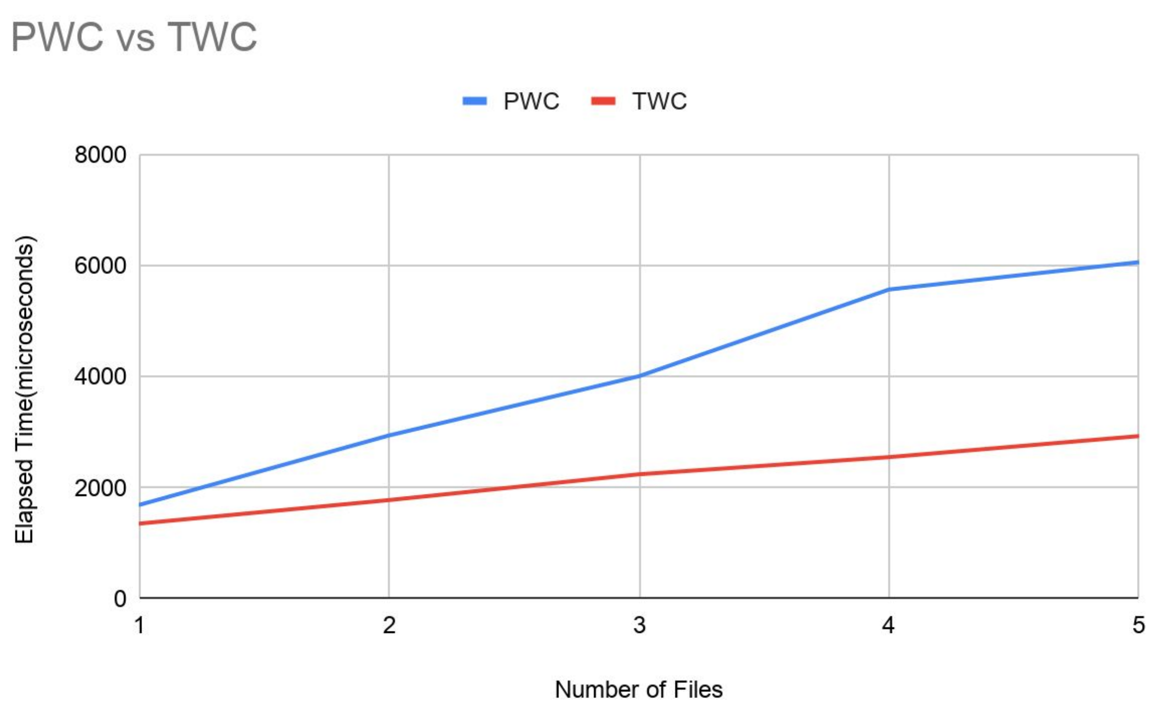
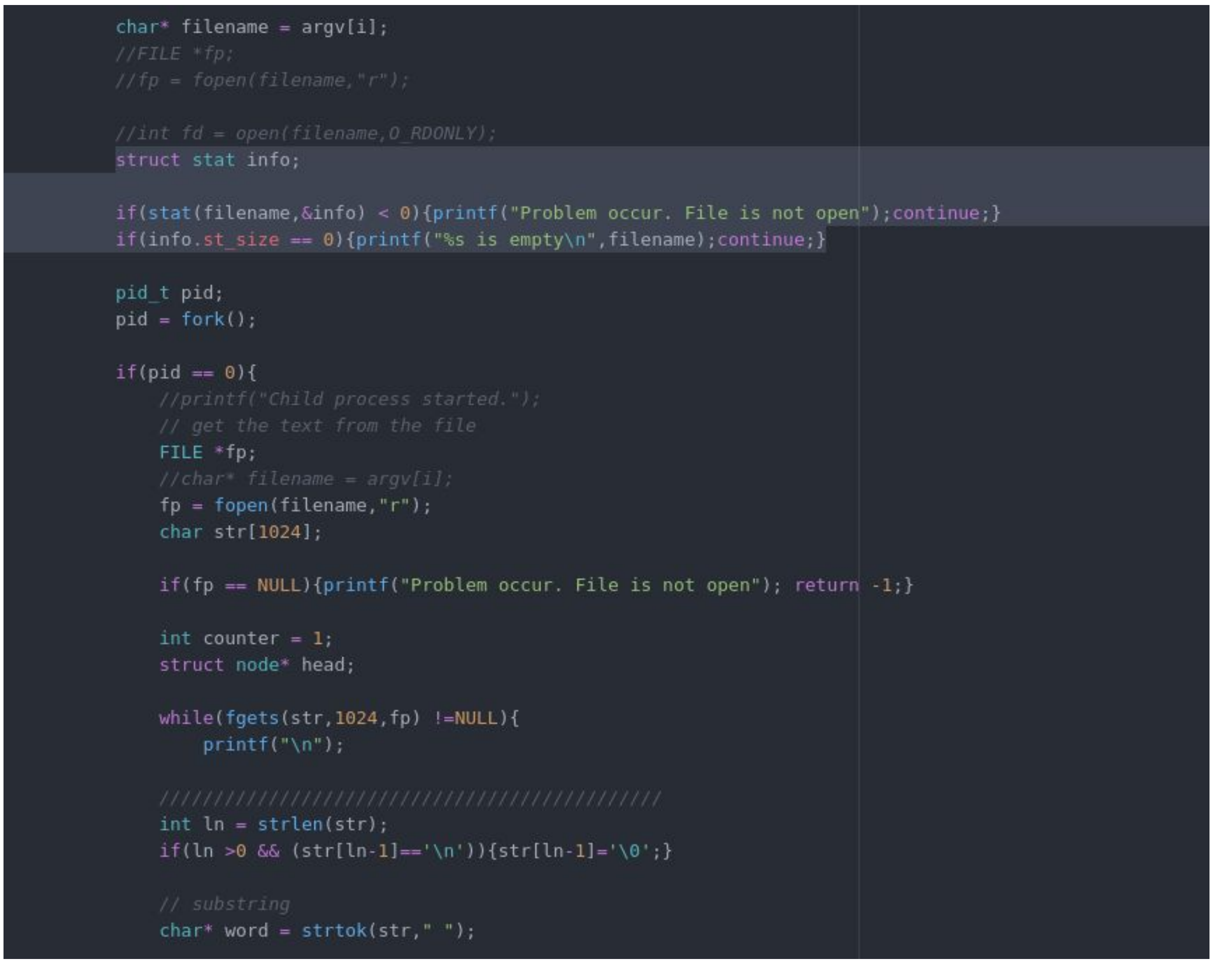
Ata Coşkun | 21503061| CS342| PROJECT 1

TIME ANALYSIS of TWC and PWC



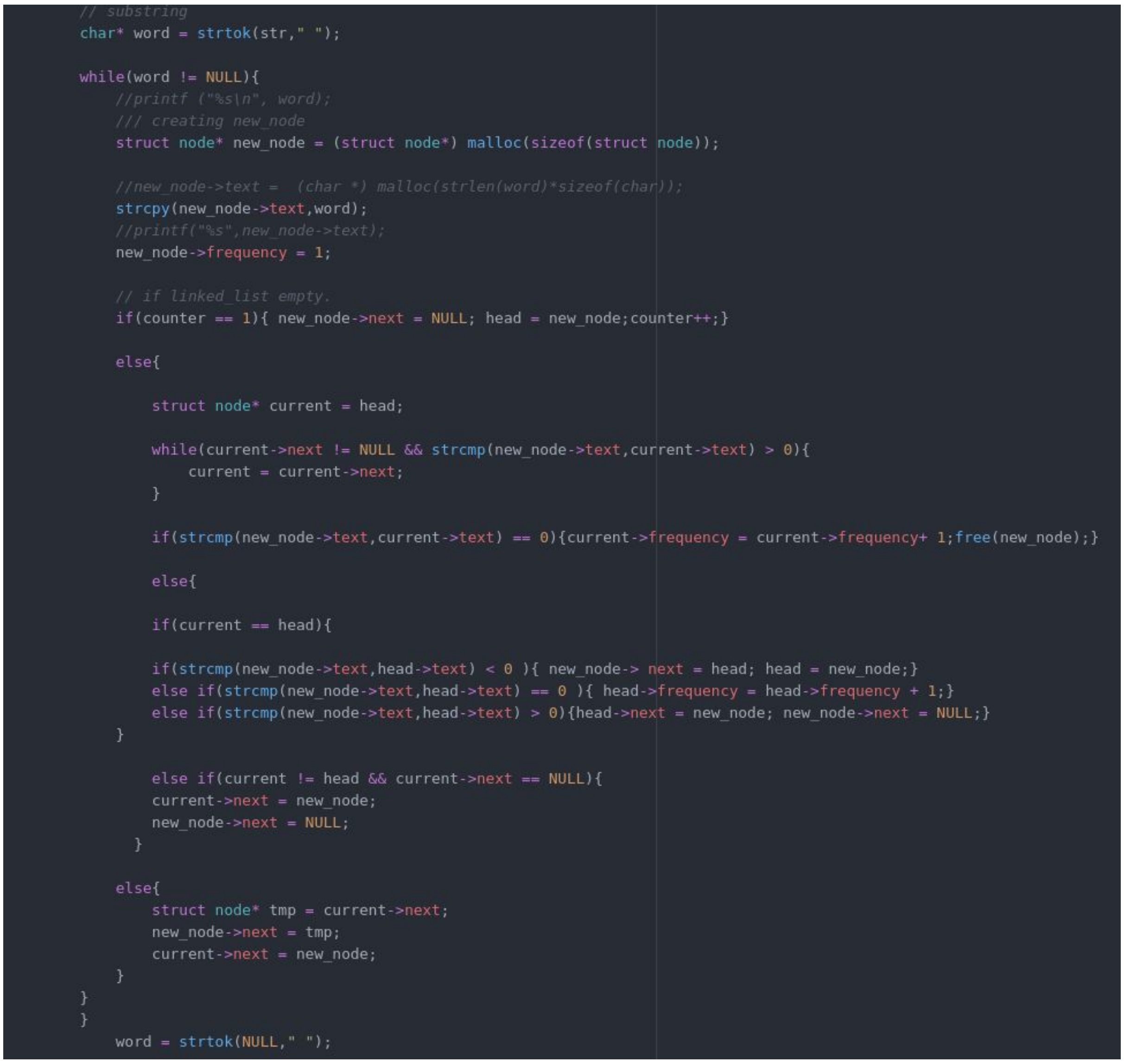
All tests are done for pwc and twc with the same files. Theoretical results and experimental results are coherent with each other. Main difference is one solution lies on the message queue approach and the other one lies on a shared memory approach. Processes are isolated. Threads use shared memory concurrently and because of that faster. Resource sharing is not expensive for threads compared to process. Creation of a thread or context switching also faster for threads. This project we use message queues. It means that there are many extra kernel interventions(system calls like mq_open, mq_send, mq_recieve) in the pwc program. When the number of inputs is increasing, growth of time elapsed function in pwc is bigger because the cost of creating processes, kernel interventions, sharing data processes are also increasing. In addition, termination cost is also less in threads.

Appendix PWC



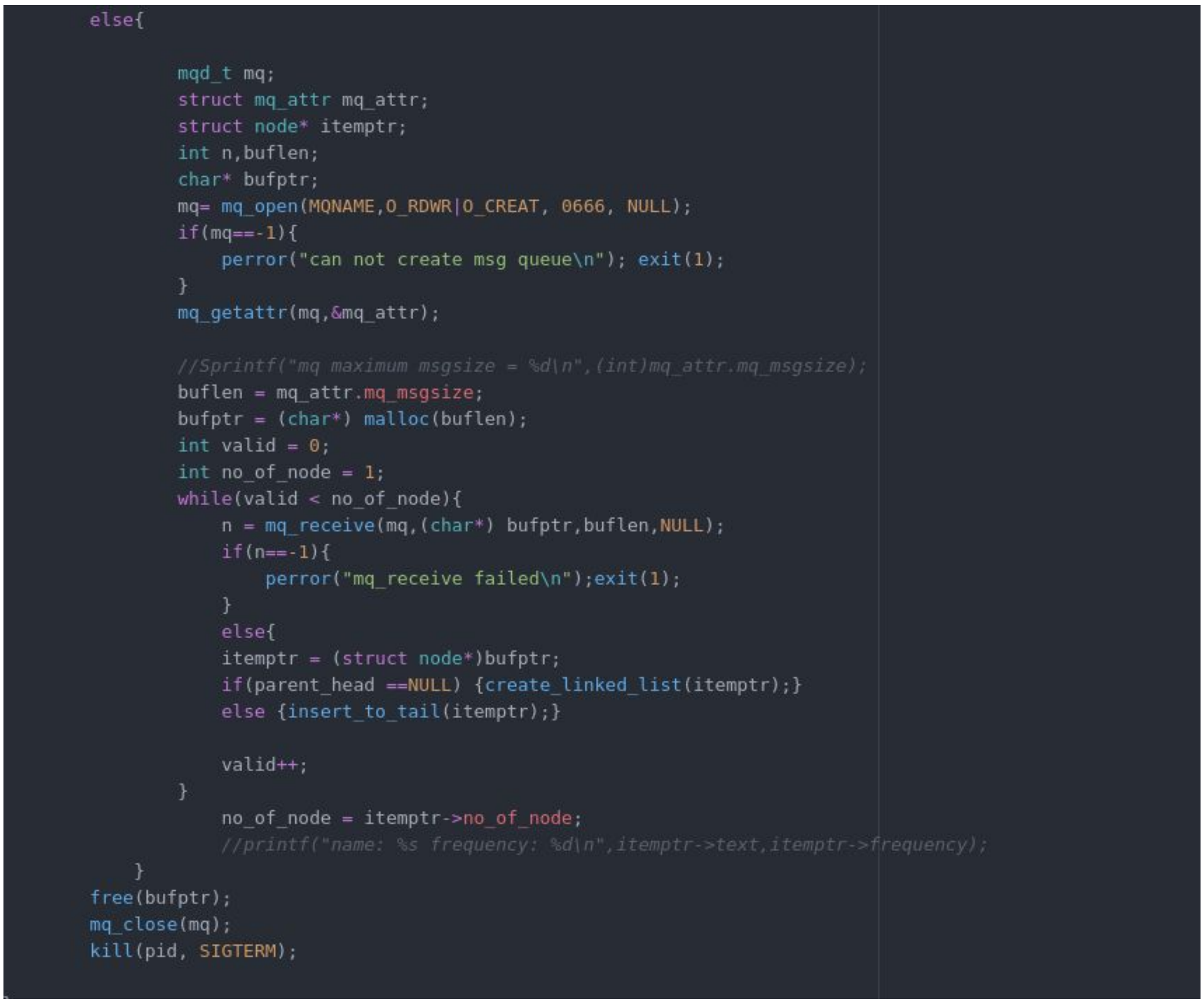
```
        char* filename = argv[i];
        //FILE *fp;
        //fp = fopen(filename,"r");

        //int fd = open(filename,O_RDONLY);
        struct stat info;

        if(stat(filename,&info) < 0){printf("Problem occur. File is not open");continue;}
        if(info.st_size == 0){printf("%s is empty\n",filename);continue;}

        pid_t pid;
        pid = fork();

        if(pid == 0){
            //printf("Child process started.");
            // get the text from the file
            FILE *fp;
            //char* filename = argv[i];
            fp = fopen(filename,"r");
            char str[1024];

            if(fp == NULL){printf("Problem occur. File is not open"); return -1;}

            int counter = 1;
            struct node* head;

            while(fgets(str,1024,fp) !=NULL){
                printf("\n");

            ///////////////////////////////////////////////
            int ln = strlen(str);
            if(ln >0 && (str[ln-1]=='\n')){str[ln-1]='\0';}

            // substring
            char* word = strtok(str," ");
```

This checks whether there are any problematic folders or empty folders and skips them. After that it starts the children's processes. In children processes I used the strtok method in C to tokenize the lines.



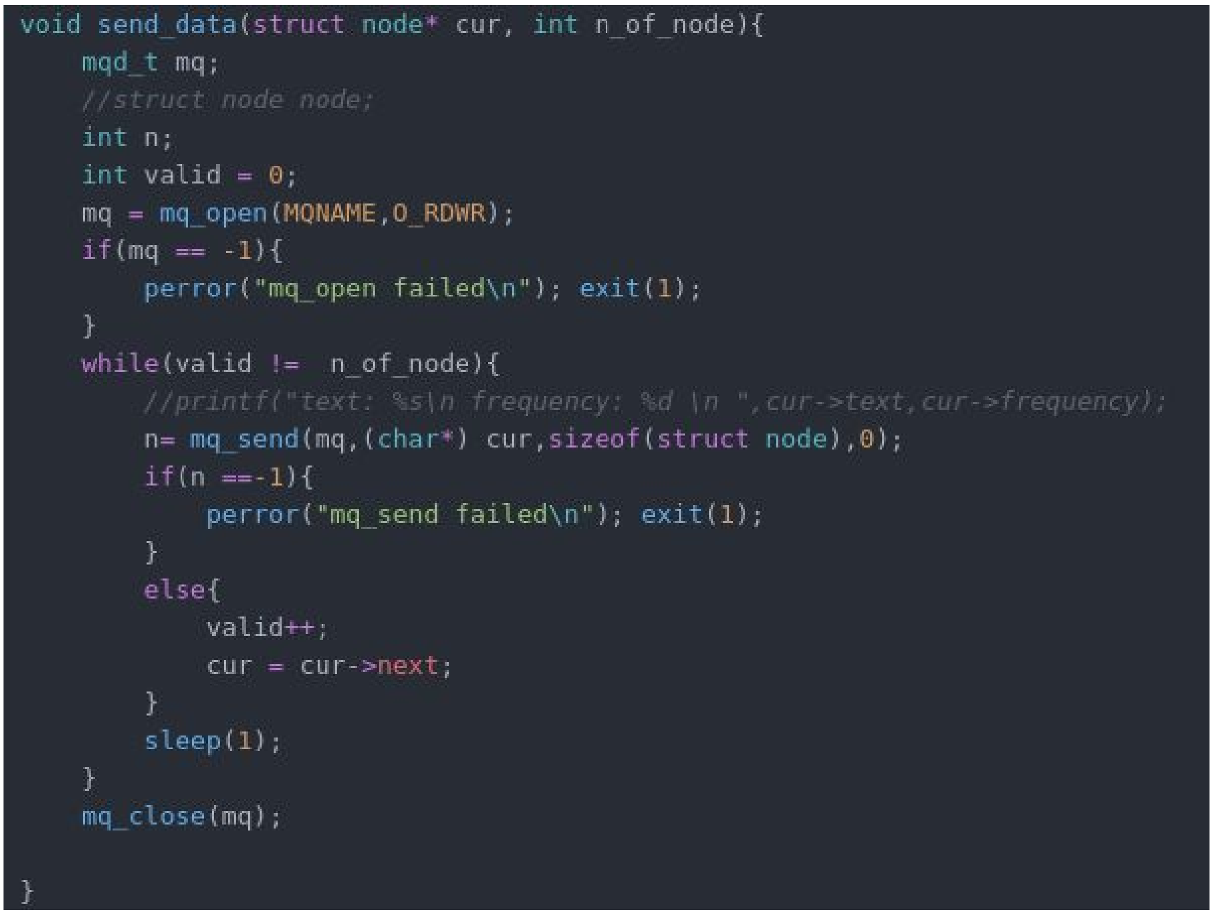
```
        // substring
        char* word = strtok(str," ");

        while(word != NULL){
            //printf ("%s\n", word);
            /// creating new_node
            struct node* new_node = (struct node*) malloc(sizeof(struct node));

            //new_node->text =  (char *) malloc(strlen(word)*sizeof(char));
            strcpy(new_node->text,word);
            //printf("%s",new_node->text);
            new_node->frequency = 1;

            // if linked_list empty.
            if(counter == 1){ new_node->next = NULL; head = new_node;counter++;}

            else{

                struct node* current = head;

                while(current->next != NULL && strcmp(new_node->text,current->text) > 0){
                    current = current->next;
                }

                if(strcmp(new_node->text,current->text) == 0){current->frequency = current->frequency+ 1;free(new_node);}

                else{

                if(current == head){

                if(strcmp(new_node->text,head->text) < 0 ){ new_node-> next = head; head = new_node;}
                else if(strcmp(new_node->text,head->text) == 0 ){ head->frequency = head->frequency + 1;}
                else if(strcmp(new_node->text,head->text) > 0){head->next = new_node; new_node->next = NULL;}
            }

                else if(current != head && current->next == NULL){
                current->next = new_node;
                new_node->next = NULL;
              }

            else{
                struct node* tmp = current->next;
                new_node->next = tmp;
                current->next = new_node;
            }
        }
        }
            word = strtok(NULL," ");
```

This block creates the nodes and constructs a sorted linked list in children. It considers insertions to empty list, head, middle and tail. While it is adding, it considers the strcmp results of strings and puts new item to appropriate sorted position.

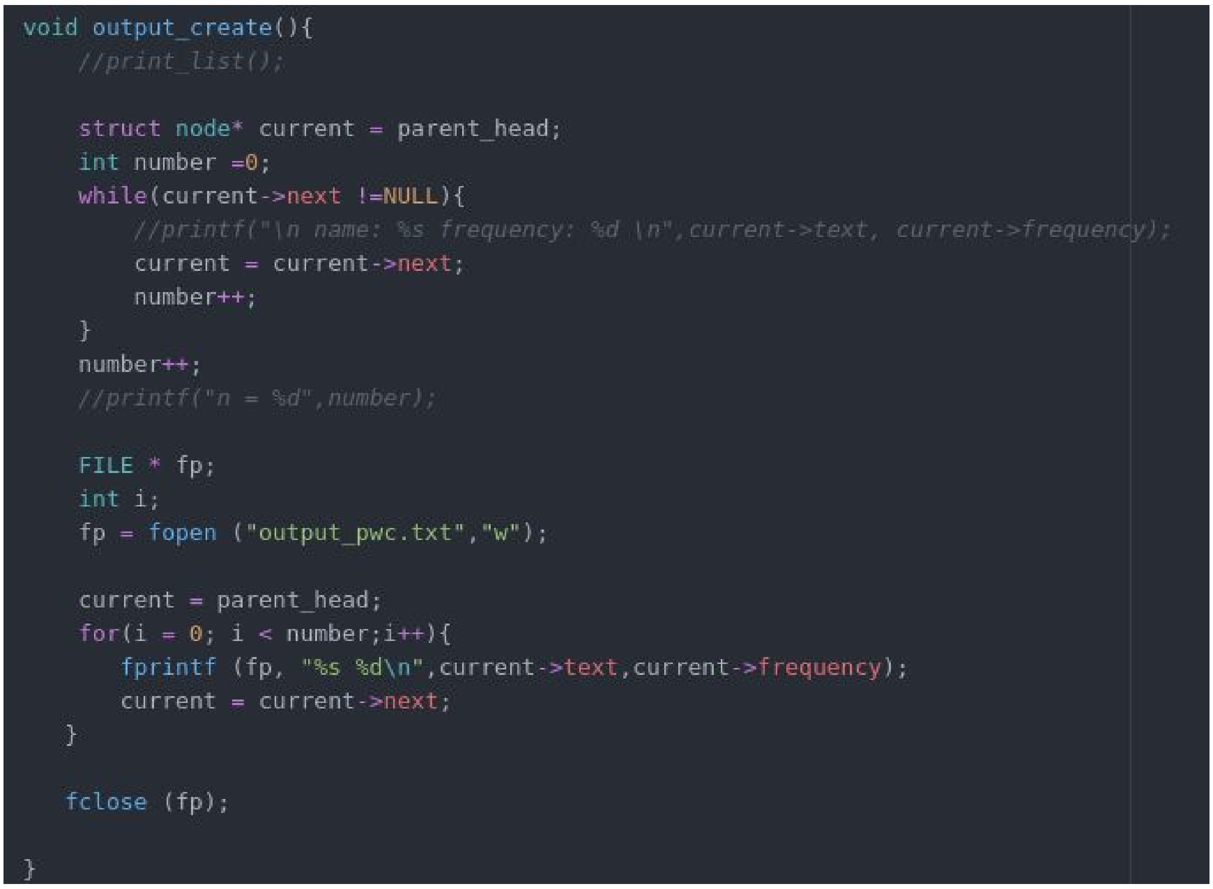


```
    else{

            mqd_t mq;
            struct mq_attr mq_attr;
            struct node* itemptr;
            int n,buflen;
            char* bufptr;
            mq= mq_open(MQNAME,O_RDWR|O_CREAT, 0666, NULL);
            if(mq==-1){
                perror("can not create msg queue\n"); exit(1);
            }
            mq_getattr(mq,&mq_attr);

            //Sprintf("mq maximum msgsize = %d\n",(int)mq_attr.mq_msgsize);
            buflen = mq_attr.mq_msgsize;
            bufptr = (char*) malloc(buflen);
            int valid = 0;
            int no_of_node = 1;
            while(valid < no_of_node){
                n = mq_receive(mq,(char*) bufptr,buflen,NULL);
                if(n==-1){
                    perror("mq_receive failed\n");exit(1);
                }
                else{
                itemptr = (struct node*)bufptr;
                if(parent_head ==NULL) {create_linked_list(itemptr);}
                else {insert_to_tail(itemptr);}

                valid++;
            }
                no_of_node = itemptr->no_of_node;
                //printf("name: %s frequency: %d\n",itemptr->text,itemptr->frequency);
        }
    free(bufptr);
    mq_close(mq);
    kill(pid, SIGTERM);
```

This block gets the nodes from the children and constructs a final linked list. It is almost the same with course slides.


```c
void send_data(struct node* cur, int n_of_node){
    mqd_t mq;
    //struct node node;
    int n;
    int valid = 0;
    mq = mq_open(MQNAME,O_RDWR);
    if(mq == -1){
        perror("mq_open failed\n"); exit(1);
    }
    while(valid !=  n_of_node){
        //printf("text: %s\n frequency: %d \n ",cur->text,cur->frequency);
        n= mq_send(mq,(char*) cur,sizeof(struct node),0);
        if(n ==-1){
            perror("mq_send failed\n"); exit(1);
        }
        else{
            valid++;
            cur = cur->next;
        }
        sleep(1);
    }
    mq_close(mq);

}
```

This block shows how children process send nodes to the parent process. I also looked at the course slide to write this part.

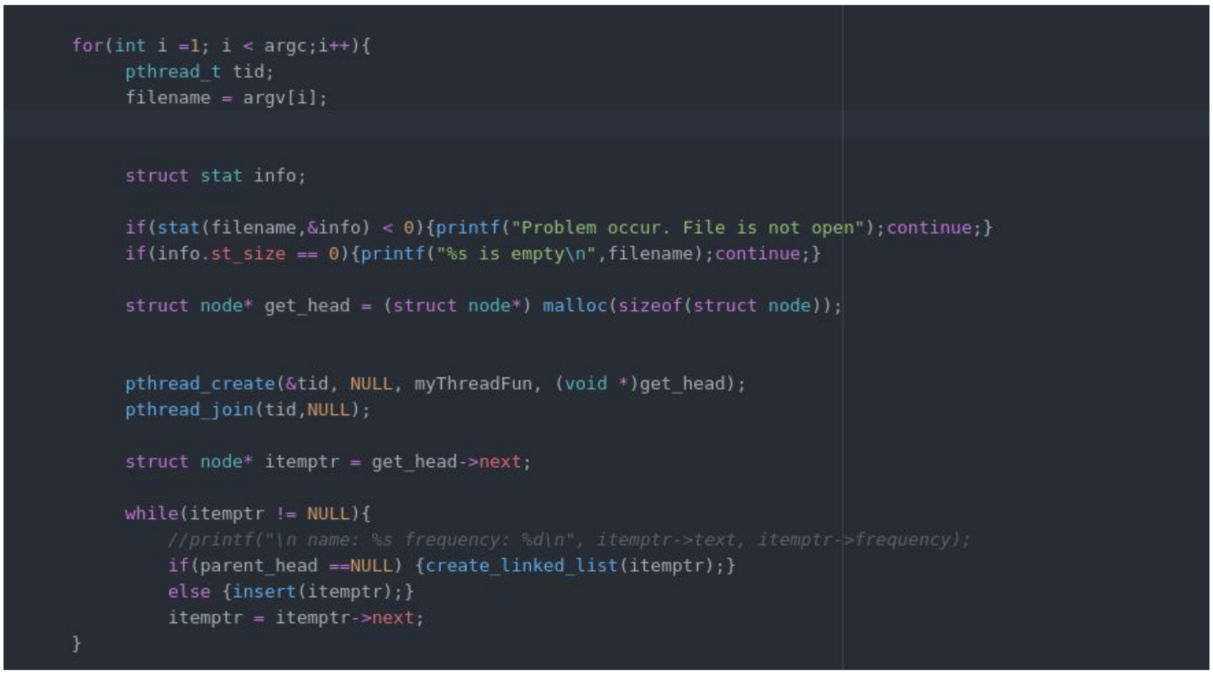


```
void output_create(){
    //print_list();

    struct node* current = parent_head;
    int number =0;
    while(current->next !=NULL){
        //printf("\n name: %s frequency: %d \n",current->text, current->frequency);
        current = current->next;
        number++;
    }
    number++;
    //printf("n = %d",number);

    FILE * fp;
    int i;
    fp = fopen ("output_pwc.txt","w");

    current = parent_head;
    for(i = 0; i < number;i++){
       fprintf (fp, "%s %d\n",current->text,current->frequency);
       current = current->next;
   }

   fclose (fp);

}
```

As a final step, this block shows how parent processes create output txt. For PWC, output file name output_pwc.txt. For TWC, output file name output_twc.txt.

Appendix TWC

In this part most of the code is parallel to PWC. Worker threads work like children processes.



```
for(int i =1; i < argc;i++){
    pthread_t tid;
    filename = argv[i];


    struct stat info;

    if(stat(filename,&info) < 0){printf("Problem occur. File is not open");continue;}
    if(info.st_size == 0){printf("%s is empty\n",filename);continue;}

    struct node* get_head = (struct node*) malloc(sizeof(struct node));


    pthread_create(&tid, NULL, myThreadFun, (void *)get_head);
    pthread_join(tid,NULL);

    struct node* itemptr = get_head->next;

    while(itemptr != NULL){
        //printf("\n name: %s frequency: %d\n", itemptr->text, itemptr->frequency);
        if(parent_head ==NULL) {create_linked_list(itemptr);}
        else {insert(itemptr);}
        itemptr = itemptr->next;
}
```

This block shows creation threads and how the main thread constructs the final linked list.

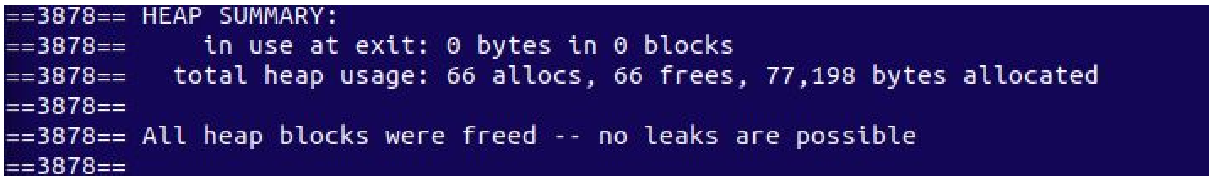
All output was fine in my tests.

There is no memory leak for both programs.

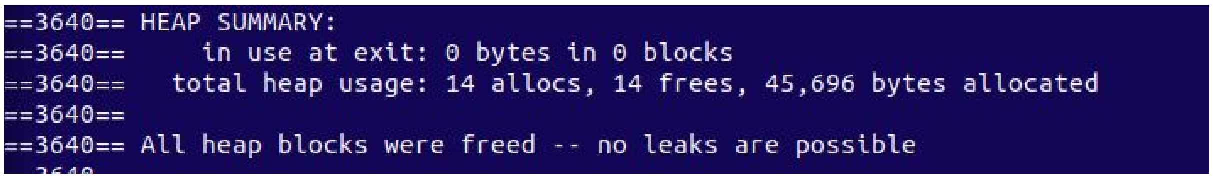
give the arguments like this : ./pwc f1.txt f2.txt f3.txt f4.txt f5.txt

./twc f1.txt f2.txt f3.txt f4.txt f5.txt

Please mail me if you have any problems: atacoskun7196@gmail.com



```
==3878== HEAP SUMMARY:
==3878==     in use at exit: 0 bytes in 0 blocks
==3878==   total heap usage: 66 allocs, 66 frees, 77,198 bytes allocated
==3878==
==3878== All heap blocks were freed -- no leaks are possible
==3878==
```



```
==3640== HEAP SUMMARY:
==3640==     in use at exit: 0 bytes in 0 blocks
==3640==   total heap usage: 14 allocs, 14 frees, 45,696 bytes allocated
==3640==
==3640== All heap blocks were freed -- no leaks are possible
```