**Московский Авиационный Институт**

**(Национальный Исследовательский Университет)**

**Факультет информационных технологий и прикладной математики**

**Кафедра вычислительной математики и программирования**

**Лабораторная работа №3 по курсу**

**«Операционные системы»**

**УПРАВЛЕНИЕ ПОТОКАМИ В ОС UNIX**

**Студент: Баранников Степан Алексеевич**

**Группа: М80–201Б-18**

**Вариант: 03**

**Преподаватель: Соколов Андрей Алексеевич**

**Оценка: \_\_\_\_\_\_\_\_\_\_\_**

**Дата: \_\_\_\_\_\_\_\_\_\_\_**

**Подпись: \_\_\_\_\_\_\_\_\_\_\_**

**Москва, 2019**

**1.** **Постановка задачи**

Сортировка слиянием, работающая в потоках.

**2.Листинг программы**

|  |
| --- |
| #include <stdio.h> #include <stdlib.h> #include <string.h> #include <unistd.h>  #include <pthread.h>  #define ERROR\_CREATE\_THREAD -11 #define ERROR\_JOIN\_THREAD -12 #define SUCCESS 1 #define FAIL 11  typedef struct thread\_data{  char \*\*line;  int start;  int end; } thread\_data;  void \*mergeSort(void \*args){  thread\_data \*data = (thread\_data\*) args;  int start = data->start, end = data->end;  printf("CATCHargs: %d, %d\n", start, end);  int status, status\_addr;  if(end - start < 2)  return (void \*) SUCCESS;  if(end - start == 2){  char \*temp;  if(strcmp(data->line[start], data->line[start+1]) > 0){  temp = data->line[start];  data->line[start] = data->line[start + 1];  data->line[start + 1] = temp;  }  return (void \*) SUCCESS;  }  pthread\_t thread\_l, thread\_r;  thread\_data \*data\_l, \*data\_r;   data\_l = data;  data\_l->start = start;  data\_l->end = start + (end-start)/2;  status = pthread\_create(&thread\_l, NULL, mergeSort,(void \*) data\_l);  if(status == 0){  pthread\_join(thread\_l,(void \*\*) &status\_addr);  if(status\_addr!=SUCCESS){  printf("ERROR");  return (void \*) FAIL;  }  }   data\_r = data;  data\_r->start = start + (end-start)/2;  data\_r->end = end;  status = pthread\_create(&thread\_r, NULL, mergeSort,(void \*) data\_r);  if (status == 0){  pthread\_join(thread\_r,(void \*\*) &status\_addr);  if(status\_addr!=SUCCESS){  printf("ERROR");  return (void \*) FAIL;  }  }  char \*\*b = (char\*\*) malloc((end - start) \* sizeof(char\*));  int b1 = start, e1 = start + (end - start)/2, b2 = e1;  for(int i = 0; i < (end - start); i++){  b[i] = (char\*) malloc(200 \* sizeof(char));  if(b1 >= e1 || (b2 < end && (strcmp(data->line[b1],data->line[b2]) > 0 || strcmp(data->line[b1],data->line[b2]) == 0))){  b[i] = data->line[b2];  b2++;  }  else{  b[i] = data->line[b1];  b1++;  }  }  for(int i = start; i < end; ++i){  data->line[i] = b[i - start];  }   return (void \*) SUCCESS; }    int main(int argc, char \*argv[]){  FILE \*datafile;  if(argc == 2){  printf("Using given file '%s'\n", argv[1]);  if(!(datafile = fopen(argv[1], "r"))){  printf("Couldn't use file '%s'\nExit...\n", argv[1]);  return 0;  }  }  else if(argc == 1){  printf("Using standart file 'data.txt'\n");  if(!(datafile = fopen("data.txt", "r"))){  printf("Couldn't use file 'data.txt'\nExit...\n");  return 0;  }  }  else{  printf("Error: you must give only 1 file!\n");  return 0;  }  int i, j;  int n;  fscanf(datafile, "%d\n", &n);  printf("\n");  char \*\*line = (char\*\*) malloc(n \* sizeof(char\*));  for(i = 0; i < n; i++){  line[i] = (char\*) malloc(200 \* sizeof(char));  fgets(line[i], 200, datafile);  printf("%d) %s", i, line[i]);  if(feof(datafile)){break;}  }  printf("\n\n");  /////////////////////////////////////////////////////   thread\_data data = {line, 0, 10};  pthread\_t thread;  int status, status\_addr;    status = pthread\_create(&thread, NULL, mergeSort, (void \*) &data);  if(status != 0) {  printf("main error: can't create thread, status = %d\n", status);  exit(ERROR\_CREATE\_THREAD);  }  else if(status == 0){  status = pthread\_join(thread, (void \*\*) &status\_addr);  if(status\_addr != SUCCESS){  printf("ERROR\n");  return 0;  }  }  printf("Hello from main!\n");    printf("\n\n");  //mergeSort\_out(line, 0, 10);  printf("\n\n");  for (i = 0; i < n; i++){  printf("%d) %s", i, line[i]);  }  for(i = 0; i < n; i++){  free(line[i]);  }  free(line);   } |

**3. Вывод**

Благодаря потокам можно распараллелить решение тех или иных рекурсивных или объемных задач, что сокращает время работы программы, то есть добавляет очень много эффективности нашим проектам.