Universidade Fernando Pessoa

Sistemas Operativos

Trabalho Prático

2019/2020

Patricia Rodrigues

35365

Índice

Objetivos

- Aplicar os conhecimentos aprendidos nas aulas de Sistemas Operativos.
- Criar um programa que consiga ordenar um grande conjunto de dados com recurso a computação paralela.

Fase 1:Ponto 1

(10%)Ler e armazenar o conjunto de valores a ordenar.

- Criar ficheiro com 100 números aleatórios entre 0 e 50 para começar:

```
void createFileInts()
{
    int * a = newIntArray(TAM);
    int i;
    uniformArray(a,TAM,0,50);
    printArray(a,TAM);
    writeInts("MergeSort.txt",a,TAM);
}
```

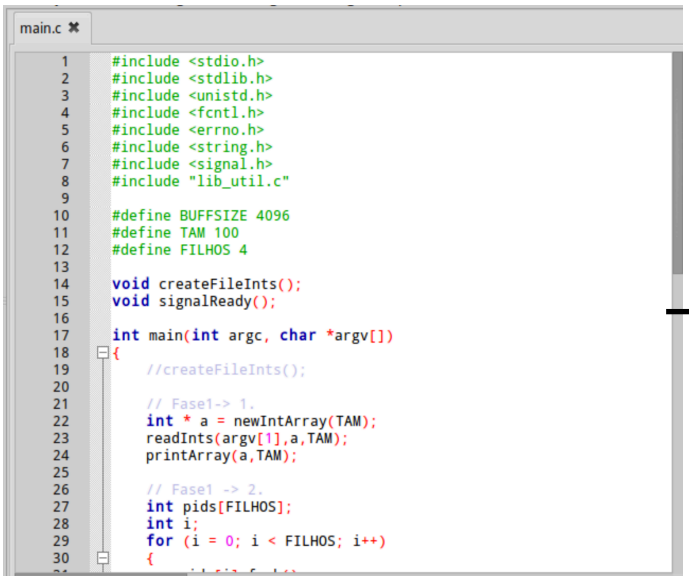
- Ler o ficheiro com os inteiros a partir do argv e armazenar num array de inteiros utilizando funções auxiliares fornecidas pelos professores:

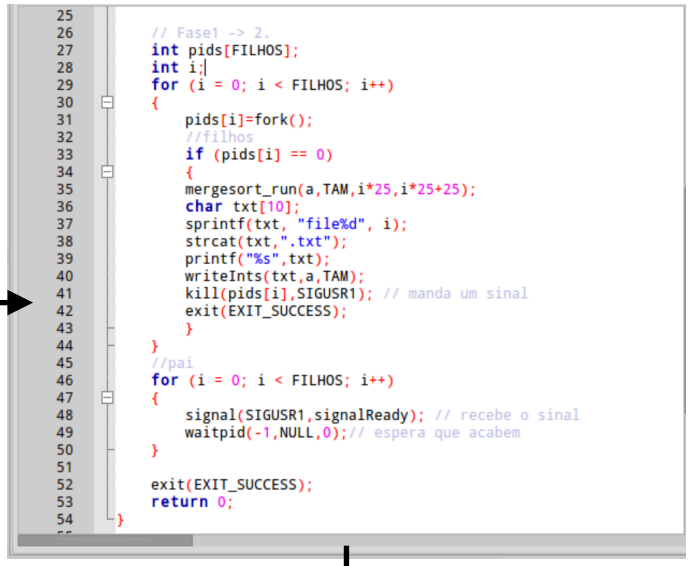
```
// Fase1-> 1.
int * a = newIntArray(TAM);
readInts(argv[1],a,TAM);
printArray(a,TAM);
```

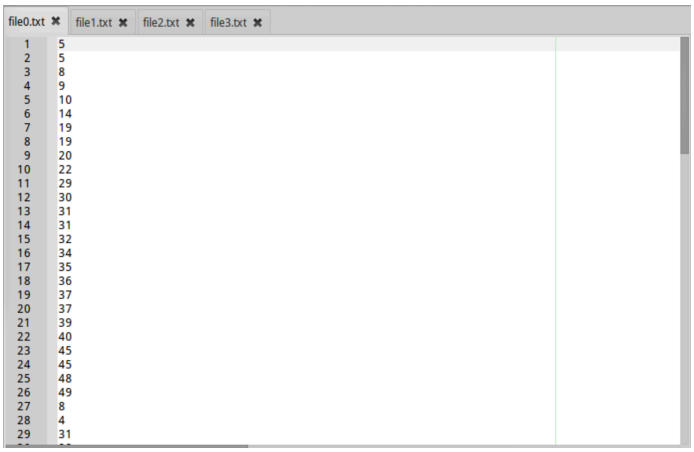
Fase 1:Ponto 2

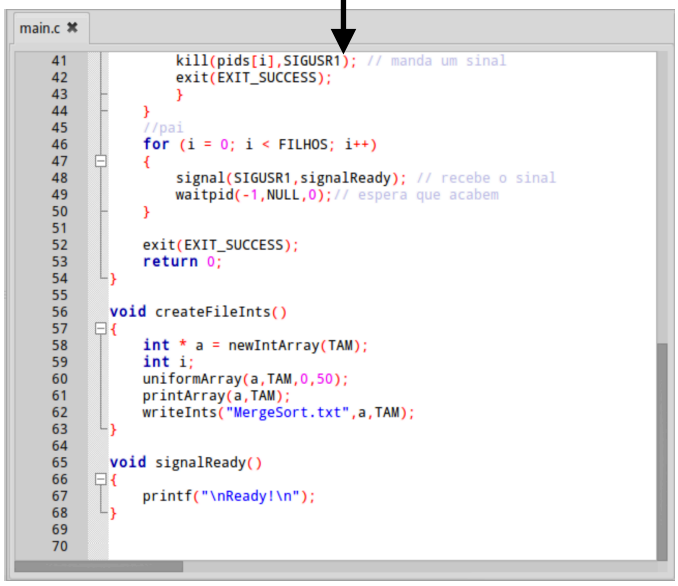
(25%) Criar N filhos, cada um deles deve ordenar a sua subsequência. Após a ordenação deve criar um ficheiro onde vai escrever a subsequência ordenada.No final o processo filho deve enviar um sinal ao pai SIGURS1 a notificar que a sua subsequência já se encontra ordenada. O pai deve esperar pela finalização dos seus filhos.

- O primeiro filho ao ser criado vai fazer o mergeSort que foi fornecido começando no seu i*25(0*25 = 0) até i*25 +25 (0*25 +25 = 25). Neste caso utilizo o 25 porque sei que só tem 100 valores mas mais para a frente vai ser atualizado para fazer a conta consoante o números de inteiros no ficheiro. Depois guarda o seu array com a sua porção ordenada num ficheiro de texto com o numero do i (file0.txt) e manda o sinal ao pai a dizer que está pronto o ficheiro. Assim como este os outros filhos fazem o mesmo com as suas porções.









Fase 1:Ponto 3

(35%) Esta etapa implica que o programa permita suportar a comunicação entre processos com recurso a pipes. Cada filho deve retornar a sua subsequência ordenada para o pai através do pipe. O programa deve suportar um protocolo de comunicação que permita ao pai saber que filho é que ordenou determinada subsequência e a que intervalo pertence. O pai recebe as subsequências ordenadas e guarda as subsequências no array original. Nesta etapa o programa deve fazer recurso da função readn e writen

- Este ponto não correu tão bem como esperava. Consegui que o pai lesse toda a informação que os Fihos mandavam pelo pipe mas depois não consegui por a dar a conversão de string para int dos valores lidos. Segue se na mesma o código que implementei.

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>
#include <string.h>
#include <signal.h>
#include "lib_util.c"


#define TAM 100
#define FILHOS 4

void createFileInts(void);
void signalReady(void);
void codigofilhos(int *a,int salto,int i,int fds[2]);
int * saveSeq(int * a,char aux[TAM]);
int stringToInt(char a[]) ;

int main(int argc, char *argv[])
{
    //createFileInts();

    // Fase1-> 1.
    int * a = newIntArray(TAM);
    readInts(argv[1],a,TAM);
    //printArray(a,TAM);

    // Fase1 -> 2.
    int pids[FILHOS];
    int i;
    // fase 1 -> 3.
    int fds[2], salto;
    pipe(fds);
    salto = TAM / FILHOS; // numero que cada filho ordena
    for (i = 0; i < FILHOS; i++)
    {
        pids[i]=fork();
        //primeiros filhos
        if (pids[i] == 0)
        {
            dup2(fds[1],1);// para escrever no pipe
            close(fds[0]);
            close(fds[1]);
            close(0);
            codigofilhos(a,salto,i,fds);
        }
    }
    //pai
    dup2(fds[0],0);//para ler do pipe
    for (i = 0; i < FILHOS; i++)
    {
        char aux[TAM] = " ";
        read(0,aux,TAM);
        write(1,aux,TAM);
        saveSeq(a,aux);
        waitpid(pids[i],NULL,0);// espera que acabem
    }
    printArray(a,TAM);
    freeIntArray(a);
    exit(EXIT_SUCCESS);
    return 0;
}

void createFileInts()
{
    int * a = newIntArray(TAM);
    uniformArray(a,TAM,0,50);
    printArray(a,TAM);
    writeInts("MergeSort.txt",a,TAM);
}

void signalReady()
{
    printf("\nReady!\n");
}

void codigofilhos(int *a,int salto,int i,int fds[2])
{

    int * merge = newIntArray(salto);
    int z,j;
    for(z=0,j=i*salto;j<(i*salto)+salto;j++,z++){
        merge[z]=a[j];
    }
    mergesort_run(merge,salto,0,salto-1);

    printf("%d*%d;%d*",getpid(),i*salto,(i*salto) +salto);
    printArray(merge,salto);
    free(merge);
    exit(EXIT_SUCCESS);
}

int * saveSeq(int * a,char aux[TAM])
{
    int start,end;
    aux = strtok(aux,"*;");
    aux = strtok(NULL,"*;");

    start= stringToInt(aux);
    printf("start: %d\n",start);
    aux = strtok(NULL,"*;");

    end = stringToInt(aux);
    printf("end: %d\n",end);
    aux = strtok(NULL," ");

    while ( aux!=NULL && start<end)
    {
        int num =0;
        num =stringToInt(aux);
        a[start]=num;
        aux = strtok(NULL," ");
        start++;
    }
    return a;
}

int stringToInt(char a[]) {
    int  i, len;

    int result=0;

    len = strlen(a);

    for(i=0; i<len; i++){

    result = result * 10 + ( a[i] - '0' );

    }

  return result;
}
```

```
Patricias-MBP:test faculdade$ gcc test.c -o test
Patricias-MBP:test faculdade$ ./test MergeSort.txt
73097*0;25*0 0 1 3 5 7 7 12 12 17 18 19 19 22 22 28 28 29 32 35 38 40 41 44 47
73098*25;50*0 2 3 3 4 4 4 5 5 14 14 15 16 17 18 21 22 23 26 28 36 37 37 39 43
73099*50;75*2 5 7 7 7 10 13 14 15 16 19 21 22 24 24 25 30 33 35 38 40 42 43 47 49
73100*75;100*1 2 4 5 6 6 9 14 15 16 19 20 23 24 24 24 29 29 30 30 30 32 32 47 49
Patricias-MBP:test faculdade$
```

Fase 1:Ponto 4

(30%) Esta etapa implica que o programa permita suportar a comunicação entre processos com recurso a Unix Domain Sokects. Cada filho deve estabelecer conexão com o server (pai). O pai deve atender as conexões e armazenar as subsequências ordenadas array original.

- Para evitar repetições fiz uma função (codigoPai(int * a,int salto,int filhos)) onde vai ser feita a criação do socket, a ligação com os clientes (filhos) e o armazenamento do as arrays ordenados. Também fiz uma função para a criação dos filhos (criarFilhos(int filhos,int *a) onde o pai vai fazer o fork() e depois cada filho faz a sua ordenação seguido de um exec para correr o client.c.

```c
#include <sys/un.h>
#include <sys/types.h>
#include <sys/uio.h>
#include <sys/socket.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <fcntl.h>

#define BUFFSIZE 4096
#define TAM 100
#define FILHOS 4

#define BUF_SIZE 4096
#define LISTENQ 5

char *socket_path = "/tmp/socket";

void createFileInts(void);
void signalReady(void);
void codigofilhos(int *a,int salto,int i);
int * saveSeq(int * a,char aux[TAM]);
int stringToInt(char a[]) ;
void criarFilhos(int filhos,int *a);
void codigoPai(int * a,int salto,int filhos);

int main(int argc, char *argv[])
{
    //createFileInts();
    // Fase1-> 1.
    int * a = newIntArray(TAM);
    readInts(argv[1],a,TAM);
    //printArray(a,TAM);
    // Fase1 -> 2.
    int salto = TAM / FILHOS;
    criarFilhos(FILHOS,a);
    //pai
    codigoPai(a,salto,FILHOS);
    printf("\nPrimeira Ordenação:\n");
    printArray(a, TAM);

    criarFilhos(FILHOS/2, a);
    codigoPai(a, salto*2,FILHOS/2);
    printf("\nSegunda Ordenação:\n");
    printArray(a, TAM);

    criarFilhos(1, a);
    codigoPai(a, 100,1);
    printf("\nUltima Ordenação:\n");
    printArray(a, TAM);
    return 0;
}
```

```c
void criarFilhos(int filhos,int *a){
    int pids[filhos];
    int salto,i;
    salto = TAM / filhos; // numero que cada filho ordena
    for (i = 0; i < filhos; i++)
    {
        //printf("\n Vou criar os meus filhos\n");
        pids[i]=fork();
        //primeiros filhos
        if (pids[i] == 0)
        {
            int * merge = newIntArray(salto);
               int z,j;
               for(z=0,j=i*salto;j<(i*salto)+salto;j++,z++){
                   merge[z]=a[j];
               }
               mergesort_run(merge,salto,0,salto-1);
            char txt[10];
            sprintf(txt, "%d.txt",i);
            writeInts(txt, merge, salto);
            freeIntArray(merge);
            char *args[]={"./client","",NULL};
            args[1]=txt;
            execvp(args[0],args);
        }
    }
}
```

1.txt

Documento de texto que cada filho criava

```c
void codigoPai(int * a,int salto,int filhos){
    int listenfd,connfd,fd,bytes;
       char buf[BUF_SIZE];                                          // buffer
       struct sockaddr_un channel_srv;

       if ( (listenfd = socket(AF_UNIX, SOCK_STREAM, 0)) == -1) {              // Creati
           perror("socket error");
           exit(-1);
       }

       unlink(socket_path);

       memset(&channel_srv, 0, sizeof(channel_srv));
       channel_srv.sun_family = AF_UNIX;
       strncpy(channel_srv.sun_path, socket_path, sizeof(channel_srv.sun_path)-1);

       if (bind(listenfd, (struct sockaddr*)&channel_srv, sizeof(channel_srv)) == -1) {
           perror("bind error");
           exit(-1);
       }
       if (listen(listenfd, LISTENQ) == -1) {                          // Config
           perror("listen error");
           exit(-1);
       }

                                                                       // Socket
```

```c
    int i;
    for( i=0;i<filhos;i++) {

           if ((connfd = accept(listenfd, NULL, NULL)) == -1) {
               perror("accept error");
               continue;
           }
           if((bytes=read(connfd, buf, BUF_SIZE))<=0){
               close(connfd);
           }

           fd = open(buf, O_RDONLY);
           if (fd < 0) {
               perror("Open");
               close(connfd);
           }

           printf("File accepted: --> %s <--\n",buf);
           int * aux = newIntArray(salto);
           readInts(buf,aux,salto);
           printArray(aux,salto);
       int z,j,i;
       strtok(buf, ".");
       i= stringToInt(buf);
       for(z=0,j=i*salto;j<(i*salto)+salto;j++,z++){
           a[j]=aux[z];
       }
       freeIntArray(aux);
    }
}
```

```
Patricias-MBP:test faculdade$ ./proj MergeSort.txt
File accepted: --> 0.txt <--
0 0 1 3 5 7 7 12 12 17 18 19 19 22 22 28 28 29 32 35 38 40 41 44 47
File accepted: --> 3.txt <--
1 2 4 5 6 6 9 14 15 16 19 20 23 24 24 24 29 29 30 30 30 32 32 47 49
File accepted: --> 2.txt <--
2 5 7 7 7 10 13 14 15 16 19 21 22 24 24 25 30 33 35 38 40 42 43 47 49
File accepted: --> 1.txt <--
0 2 3 3 4 4 4 5 5 14 14 15 16 17 18 21 22 23 26 28 36 37 37 39 43

Primeira Ordenação:
0 0 1 3 5 7 7 12 12 17 18 19 19 22 22 28 28 29 32 35 38 40 41 44 47 0 2 3 3 4 4 4 5 5 14 14 15
16 17 18 21 22 23 26 28 36 37 37 39 43 2 5 7 7 7 10 13 14 15 16 19 21 22 24 24 25 30 33 35 38 4
0 42 43 47 49 1 2 4 5 6 6 9 14 15 16 19 20 23 24 24 24 29 29 30 30 30 32 32 47 49
File accepted: --> 0.txt <--
0 0 0 1 2 3 3 3 4 4 4 5 5 5 7 7 12 12 14 14 15 16 17 17 18 18 19 19 21 22 22 22 23 26 28 28 28
29 32 35 36 37 37 38 39 40 41 43 44 47
File accepted: --> 1.txt <--
1 2 2 4 5 5 6 6 7 7 7 9 10 13 14 14 15 15 16 16 19 19 20 21 22 23 24 24 24 24 24 25 29 29 30 30
 30 30 32 32 33 35 38 40 42 43 47 47 49 49

Segunda Ordenação:
0 0 0 1 2 3 3 3 4 4 4 5 5 5 7 7 12 12 14 14 15 16 17 17 18 18 19 19 21 22 22 22 23 26 28 28 28
29 32 35 36 37 37 38 39 40 41 43 44 47 1 2 2 4 5 5 6 6 7 7 7 9 10 13 14 14 15 15 16 16 19 19 20
 21 22 23 24 24 24 24 24 25 29 29 30 30 30 30 32 32 33 35 38 40 42 43 47 47 49 49
File accepted: --> 0.txt <--
0 0 0 1 1 2 2 2 3 3 3 4 4 4 4 5 5 5 5 5 6 6 7 7 7 7 7 9 10 12 12 13 14 14 14 14 15 15 15 16 16
16 17 17 18 18 19 19 19 19 20 21 21 22 22 22 22 23 23 24 24 24 24 24 25 26 28 28 28 29 29 29 30
 30 30 30 32 32 32 33 35 35 36 37 37 38 38 39 40 40 41 42 43 43 44 47 47 47 49 49

Ultima Ordenação:
0 0 0 1 1 2 2 2 3 3 3 4 4 4 4 5 5 5 5 5 6 6 7 7 7 7 7 9 10 12 12 13 14 14 14 14 15 15 15 16 16
16 17 17 18 18 19 19 19 19 20 21 21 22 22 22 22 23 23 24 24 24 24 24 25 26 28 28 28 29 29 29 30
 30 30 30 32 32 32 33 35 35 36 37 37 38 38 39 40 40 41 42 43 43 44 47 47 47 49 49
Patricias-MBP:test faculdade$
```