MultiThreaded e IPC Programação

Relatório do Trabalho Prático

Sistemas Operativos

Paul Andrew Crocker
Valderi Leithardt

Trabalho realizado por:

António Morais nº 30268
Luis Santos nº 30646
Pedro Torres nº 34663

Índice

Introdução

Este relatório surgiu no âmbito da unidade curricular Sistemas Operativos da Universidade da beira interior. Neste trabalho vamos desenvolver um programa utilizando conceitos lecionados nas aulas de sistemas operativos e estrutura de dados.

Este trabalho tem como objectivo o desenvolvimento de um programa, implementando linguagem C, utilizando uma estrutura de dados sincronizados para ser usado num servidor aplicacional e seus clientes. A comunicação entre servidor e cliente será efectuada utilizando um mecanismo de IPC.

Scripts e suas funcionalidades

Para a realização deste trabalho utilizamos funções já definidas pelo docente, implementando mais algumas para que o programa cumpra o objetivo apresentado.

Iniciamos o trabalho, criando uma lista de uma estrutura de dados onde poderemos armazenar as mensagem, e uma outra para armazenar a cabeça de lista da estrutura das mensagens.

```
typedef struct mensagem
{
    int id;
    char conteudo[32];
    struct mensagem * nseg;
    struct mensagem * nant;
} mensagem;


typedef struct nodo
{
    int id;
    struct mensagem * head;
    struct nodo * nseg;
    struct nodo * nant;
} nodo;
```

Depois das estruturas criadas, houve necessidade de implementar funções para uma fácil manipulação das mesmas.

```
nodo * criaNodo () {
    struct nodo*nv;
    nv=(nodo*)malloc(sizeof(nodo));
    nv->nseg=NULL;
    nv->nant=NULL;
    return(nv);
}

mensagem * criaMensagem () {
    struct mensagem * nm;
    nm=(mensagem*)malloc(sizeof(mensagem));
    nm->nseg=NULL;
    nm->nant=NULL;
    return (nm);
}
```

Foram definidas várias funções para a manipulação de listas, assim como a criação da mesma, visualização e eliminação. Funções que iremos utilizar para mostrar e manipular os valores das listas.

```
mensagem * deletaMensagem (mensagem * l, mensagem * r)
{
    if(r==l)
        l=l->nseg;
    if(r->nant!=NULL)
        r->nant->nseg=r->nseg;
    if(r->nseg!=NULL)
        r->nseg->nant=r->nant;
    r->nant=NULL;
    r->nseg=NULL;
    free(r);
    return (l);
}
mensagem * insereMensagemFim (mensagem * l, mensagem * nm) {
    mensagem * aux=l;
    if (l==NULL)
        return nm;
    while(aux->nseg!=NULL)
      aux=aux->nseg;

        aux->nseg=nm;
      nm->nant=aux;
      nm->nseg=NULL;


    return (l);
}
```

Após a criação das estruturas e as suas respetivas funções de manipulação, utilizamos o programa fornecido pelo o docente, relativamente ao IPC.

No ficheiro “cliente.c”, implementamos um pequeno código que nos dá instruções de como devera ser feito o nosso input.

```
 fprintf(stderr, "------------CLIENTE----------\n");
   fprintf(stderr, "Inserir -> 1,idFila,idMensagem,Conteudo\n");
   fprintf(stderr, "Eliminar -> 2,idFila,idMensagem\n");
   fprintf(stderr, "Visualizar -> 3,idFila\n");
   fprintf(stderr, "Sair -> 0\n");
```

Visto que temos que estabelecer uma comunicação entre cliente e servidor, pretendemos com que o servidor interprete o input e realize as funções ordenadas pelo utilizador. Para isto tivemos que manipular o input dividindo a string utilizando os 3 primeiros argumentos e convertendo em inteiros.

```c
int *splitbuffer(char buff[MAXBUFF],int *tamanho)
{
    int *array;
    array=(int *)malloc(sizeof(int));
    int length=0;
    char *aux;

    aux=strtok(buff, ",");
    while(aux && length != 3)
    {
        array[length]=atoi(aux);
        array=(int *)realloc(array,++length*sizeof(int));
        aux=strtok(NULL, ",");
    }

    *tamanho=length;

    return array;
}
```

Depois de interpretar os comandos fornecidos pelo utilizador, o “server.c” irá realizar as instruções submetidas por este, sendo assim necessário um ciclo while(1) para a comunicação server-cliente se mantenha ligada e suas respetivas condições.

```c
while(1)
{

//aux 2 cria uma copia do buffer

char *aux2;
aux2=(char *)malloc(MAXBUFF*sizeof(char));

/* Read the filename from the IPC descriptor */

if ((n = read(readfd, buff, MAXBUFF)) != MAXBUFF)
{

err_sys("Server: filename read error!");

}
memset(aux2,'\0',MAXBUFF);
strcpy(aux2,buff);
int tamanho=0;
int *a; //array que ira receber a string de numeros e converter em um array de int
a=splitbuffer(aux2,&tamanho);//funcao que faz um array de int com os 3 primeiros elementos da string
```

Implementamos 3 funções para a manipulação dos dados, guardando uma copia do buffer para que este não seja perdido e não alterado durante o seguimento do programa.

Usamos uma função para inserir mensagem numa lista, para visualizar assim como para eliminar uma mensagem de uma determinada lista.

```
if(a[0]==1)
{

    char *c; // string que recebe o quarto elemento do input
    c=fourthString(buff); // funcao extrai o quarto elemento
    nodo * aux = lista;

    while(aux!=NULL)
    {
        if(a[1]==aux->id)
        {
            mensagem  *novaMesg= criaMensagem();
            novaMesg->id=a[2];

            for (int i = 0; i < strlen(c) ;i++)
            {
                novaMesg->conteudo[i]=c[i];

            }


            aux->head=insereMensagemFim(aux->head,novaMesg);
            break;

        }
        aux=aux->nseg;

    }

    fprintf(stderr, "1-Comando executado\n");

  printLista(aux->head);


    memset(buff,'\0',MAXBUFF);
    strcat(buff, "Mensagem inserida\n");
    n = strlen(buff);
    write(writefd, buff, n);


}
```

Figura 7 - Inserção na lista de mensagens

Usamos uma segunda função para eliminar uma mensagem da sua respetiva lista, caso o primeiro caracter do input seja igual a 2. Devolvendo, assim, a cabeça da lista já com a mensagem eliminada.

```
if(a[0]==2)
{
    nodo *aux=lista;
    while(aux->id!=a[1])
    {
        aux=aux->nseg;
    }

    if(aux->head==NULL)
    {
        memset(buff,'\0',MAXBUFF);
        strcat(buff, "Nao tem mensagem para apagar\n");
        n=strlen(buff);
        write(writefd,buff,n);
    }

    else
    {
        mensagem * maux=aux->head;

        while(aux!=NULL)
        {
            if(maux->id==a[2])
            {
                aux->head=deletaMensagem(aux->head,maux);
                memset(buff,'\0',MAXBUFF);
                strcat(buff, "Mesagem apagada com sucesso\n");
                n = strlen(buff);
                write(writefd,buff,n);
                break;
            }

            else
            {
                maux=maux->nseg;
            }

            if(maux==NULL)
            {
                memset(buff,'\0',MAXBUFF);
                strcat(buff, "mensagem inexistente\n");
```

Criamos, por fim uma última instrução para a visualização dos id's da mensagem numa respetiva lista, designada pelo utilizador.

```
if(a[0]==3)
{
     nodo *aux=lista;
    while(aux->id!=a[1])
    {
        aux=aux->nseg;
    }

    if(aux->head==NULL)
    {
        memset(buff,'\0',MAXBUFF);
        strcat(buff, "Nao existe\n");
        n=strlen(buff);
        write(writefd,buff,n);
    }

    else
    {
        mensagem * maux=aux->head;

        if(aux!=NULL)
        {
            printListamensagem(maux);

        }

        else
        {
           maux=maux->nseg;
        }


    }

    fprintf(stderr, "3-Comando executado\n");
    memset(buff,'\0',MAXBUFF);
    strcat(buff, "Mensagem Visualizada\n");
    n = strlen(buff);
    write(writefd, buff, n);
```

O servidor, ainda permite a leitura de um ficheiro introduzido pelo utilizador, enviando ao cliente o conteúdo do mesmo. Mas esta função já estava implementada pelo docente.

Conclusão

No final deste trabalho podemos dizer que nos deparamos com diversas dificuldades, relativamente ao multithreading assim como á manipulação de listas. Não podemos dizer que alcançamos os nossos objetivos, mas podemos dizer que com o nossos trabalho preenchemos alguns dos requisitos propostos pelo docente. Foi um trabalho rigoroso, permitindo, de certa forma, o desenvolvimento das nossas capacidades quanto programadores.