PSPD – Programação para Sistemas Paralelos e Distribuídos
Prof. Fernando W Cruz

Exercícios

1. O código a seguir é uma versão não-distribuída de uma aplicação que recebe dois números como argumento e realiza as operações de soma e subtração desses números. Com base nesse código, analise a versão de Arquivo de Definição de Interface (IDF - Interface Definition File) RPC/XDR e gere uma versão RPC do referido código.

Arquivo IDF calcula.x:

```
struct operandos {
        int x;
        int y;
};

program PROG {
        version VERSAO {
                int ADD(operandos) = 1;
                int SUB(operandos) = 2;
        } = 10;
} = 555;
```

Código, versão local (loc_calcula.c):

```c
#include <stdio.h>
#include <stdlib.h>
int add (int x, int y) {
  int result;
  printf("Requisicao de adicao para  %d e %d\n", x, y);
  result = x + y;
  return(result);
} /* fim funcao add */

int sub (int x, int y) {
  int result;
  printf("Requisicao de subtracao para  %d e %d\n", x, y);
  result = x - y;
  return(result);
} /* fim funcao sub */

int main( int argc, char *argv[]) {
  int x,y;
  if (argc!=3) {
    fprintf(stderr,"Uso correto: %s num1 num2\n",argv[0]);
    exit(0); }
  /* Recupera os 2 operandos passados como argumento */
  x = atoi(argv[1]);   y = atoi(argv[2]);
  printf("%d + %d = %d\n",x,y, add(x,y));
  printf("%d - %d = %d\n",x,y, sub(x,y));
  return(0);
} /* fim main */
```

2. O código a seguir é uma versão não-distribuída de uma aplicação que recebe números como argumento e gera uma lista encadeada. Essa lista é passada para funções que percorrem a lista, imprimindo-a e calculando o resultado da soma desses números. Com base nesse código, analise a versão de Arquivo de Definição de Interface (IDF - Interface Definition File) RPC/XDR e gere uma versão RPC do referido código. O arquivo IDF somalista.x é o seguinte:

```
1   #define VERSION_NUMBER 1
2   struct lista {
3           int             num;
4           struct lista    *prox;
5   };
6   typedef struct lista lista;
7
8   program SOMALISTA_PROG {
9       version SOMALISTA_VERSION {
10          int SOMAL(lista) = 1;
11      } = 1;
12  } = 0x13;
```

Código, versão local (loc_somalista.c):

```
1   #include <stdio.h>
2   #include <stdlib.h>
3   /* lista_encadeada local */
4   struct lista {
5       int    num;
6     struct lista  *prox;
7   };
8   typedef struct lista lista;
9
10  void imprime_lista(lista *ap_lista) {
11
12    while (ap_lista) {
13      printf("%d ",ap_lista->num);
14      ap_lista =ap_lista->prox;
15    } /* fim-while */
16    printf("\n");
17  } /* fim procedure */
18
19  void imprime_soma(lista *ap_lista) {
20    int result = 0;
21    while (ap_lista) {
22        result   += ap_lista->num;
23        ap_lista  = ap_lista->prox;
24    } /* fim-while */
25    printf("A soma eh ... %d\n",result);
26  } /* fim - print_sum */
27
28  int main( int argc, char *argv[]) {
29    int     n_termos,i;
30    lista   *ap, *inicio_lista, *apaux;
31
32    if (argc<2) {
33      fprintf(stderr,"Uso correto: %s num1 num2 ...\n",argv[0]);
34      exit(0); }
35
36    n_termos = argc-1;
37    inicio_lista = (lista *) malloc(sizeof(lista));
38    ap = inicio_lista;
39    for (i=0; i<n_termos; i++) {
40        ap->num  = atoi(argv[i+1]);
41        ap->prox = (lista *) malloc(sizeof(lista));
42        apaux    = ap;
43        ap       = ap->prox;
44    } /* fim-for */
45
46    free(apaux->prox);
47    apaux->prox=NULL;
48    imprime_lista(inicio_lista);
49    imprime_soma(inicio_lista);
50    return(0);
51  } /* fim programa */
```

3. O código a seguir é uma versão não-distribuída de uma aplicação que recebe uma sequência de números como argumento e monta um vetor (dinâmico), em função dos parâmetros informados. Em seguida, o vetor é passado como parâmetro para soma(), que percorre o vetor e devolve o resultado para a função main(). Com base nesse código, analise a versão de Arquivo de Definição de Interface (IDF - Interface Definition File) RPC/XDR apresentado e gere uma versão RPC do referido código.

Arquivo IDF somavet.x:

```
/* Arquivo de definicao para um parametro de tamanho variavel */
typedef int vetint<>; /* declaracao de um vetor de inteiros */

program SOMAVET_PROG {
    version SOMAVET_VERSION {
        int SOMAV(vetint) = 1;
    } = 1;
} = 0x9;
```

Código, versão local (loc_somavet.c):

```
#include <stdio.h>
#include <stdlib.h>

int soma(int *x, int n) {
  int i, result;

  printf("Requisicao para adicao de %d numeros\n", n);
  result=0;
  for (i=0; i<n; i++) {
    result += x[i];
  } /* fim-for */
  return (result);
} /* fim vadd */

int main( int argc, char *argv[]) {
  int *ints, n_termos, i, res;

  if (argc<2) {
    fprintf(stderr,"Uso correto: %s num1 num2 ...\n",argv[0]);
    exit(0); }

    /* recupera os numeros que devem ser adicionados */
  n_termos = argc-1;
  ints = (int *) malloc(n_termos * sizeof( int ));
  if (ints==NULL) {
    fprintf(stderr,"Erro na alocacao de memoria\n");
    exit(0); }
  /* preenche a estrutura dinamica com os valores informados pelo usuario */
    for (i=1;i<argc;i++) {
      ints[i-1] = atoi(argv[i]);
    } /* fim-for */
  /* imprime o resultado da soma */
    res = soma(ints, n_termos);
    printf("%d",ints[0]);
    for (i=1; i<n_termos; i++) {
      printf(" + %d",ints[i]);
    } /* fim-for */
    printf(" = %d\n",res);
    return(0);
  } /* fim programa principal */
```