TD1 Prog Distr

Exercice 1 : grammaire RPCL

a) Quelles sont les règles servant de “point d’entrée” de la grammaire RPCL ?
   - program-def → service RPC
   - spécification → paramètres + valeurs de retours

b) Quels sont les symboles non-terminaux non définis dans l’extrait de l’annexe ?
   - identifier
   - decimal-constant
   - hexadecimal-constant
   - octal-constant

   ⇒ cf le texte RFC

c) Quelles règles utiliser pour spécifier la valeur de retour et le ou les paramètres ?
   - valeur de retour : règle “proc-return”
   - paramètres :
     - règle “procedure-def”
     - règle “proc-firstarg”

Exercice 2 : mon premier fichier de spécification RPCL

Soit le fichier calcul.x de spécification RPCL suivant :

```
unsigned int data;

struct {
	unsigned int reponse;
};

typedef struct reponse reponse;

program SIGN {
	version VERSION_UN {
		reponse SIGN(data) = 1; /* 1 == positive data; 0 ==
negative data */
	} = 1;
} = 0x20000001;
```

a) Ce fichier est-il correct syntaxiquement ? Corrigez les éventuelles erreurs.
   - Non ce fichier n’est pas correct syntaxiquement. Voici une correction des erreurs :

```
typedef unsigned int data;

struct reponse{
	unsigned int reponse;
};

typedef struct reponse reponse;

program SIGN {
	version VERSION_UN {
		reponse SIGN(data) = 1;
	} = 1;
} = 0x20000001;
```

b) Comment réécrire plus simplement cette spécification ?

```
program SIGN {
	version VERSION_UN {
		unsigned SIGN(unsigned int) = 1;
	} = 1;
} = 0x20000001;
```

Exercice 3 : fichier de spécification RPCL et code distant

Soit le fichier calcul.x de spécification RPCL suivant :

```
struct {
	unsigned int arg1;
	unsigned int arg2;
};

typedef struct data data;

struct reponse {
	unsigned somme;
	int errno;
}

typedef struct reponse;

program CALCUL {
     version VERSION_UN {
```

```
          void CALCUL_NULL() = 0;
          reponse CALCUL_ADDITION(data) = 0;
     } = 1;
} = 0x2000001;
```

a) Vérifiez la syntaxe de cette spécification RCPL et corrigez les erreurs éventuelles.

```
struct data {
      unsigned int arg1;
      unsigned int arg2;
};

typedef struct data data;

struct reponse {
      unsigned int somme;
      int errno;
};

typedef struct reponse reponse;

program CALCUL {
     version VERSION_UN {
          void CALCUL_NULL(void) = 0;
          reponse CALCUL_ADDITION(data) = 1;
     } = 1;
} = 0x2000001;
```

b) Complétez le code des fonctions distantes suivantes (squelettes générés par rpcgen):

```
#include "calcul.h"

void * calcul_null_1_svc (void *argp, struct svc_req *rqstp) {
      static char* result;

      /*
       * insert server code here
      */

      result = NULL;

      return ((void *) &result);
}
```

```
reponse * calcul_addition_1_svc (data * argp, struct svc_req *rqstp) {
	static reponse result;

	/*
	* insert server code here
	*/

	result.somme = argp→arg1 + argp→arg2;
	unsigned int max = argp→arg1 > argp→arg2?
			argp→arg1 : argp→arg2;
	if (max > result.somme)
		result.errno = 1;
	else
		result.errno = 0;

	return (&result);
}
```

Exercice 4 : écrire une spécification RPCL ; tableau de taille variable

Vous allez écrire le fichier de spécification répondant au besoin suivant : une procédure distante de calcul de la valeur moyenne d'une série de valeurs stockées dans un tableau de "double". La valeur retournée sera un "double".

a) Écrivez une spécification RCPL avec un tableau de taille fixe de 200. Prenez en compte le fait que le nombre de valeurs stockées peut être inférieur à 200. Comment sera implémenté en C ce tableau spécifié en RCPL (cf. Annexe 1) ?

```
struct input_data {
	double input_data[200];
	int input _data len;
};

typedef struct input_data input_data;

program AVERAGE_PROG {
	version VERSION_ONE {
		double AVERAGE (input_data) = 1;
	} = 1;
} = 0x20000001;
```

```
struct input_data {
	double input_data[200];
	int input_data len;
};
```

```
typedef struct input_data input_data;
```

b) Écrivez une spécification RCPL avec un tableau de taille variable et maximale de 200.

```
struct input_data {
      double input_data<200>;
};
...
```

⇒ génération par rpcgen

c) En tenant compte d ela manière d'implémenter les tableaux RCPL en langage C (cf. Annexe 1), donnez le code en langage C de la structure de données correspondantes à la spécification précédente.

```
struct input_data {
      double * input_data;
      int input_data len;
};
...
```

⇒ génération par rpcgen

d) Comment sera implémentée la spécification d'un tableau de taille variable dont la taille maximale n'est pas indiquée ?

```
struct input_data {
      double input_data<>;
};
...
```

⇒ Après rpcgen

```
struct input_data {
      double * input_data;
      int input_data len;
};
...
```