ESERCITAZIONE 8
Remote Procedure Call

- Bernardi Daniel
- Chichifoi Karina
- Gjura Endri
- Ivan Andrei Daniel
- Pizzini Cavagna Hiari

Introduzione

Il Client chiede ciclicamente di scegliere la funzione da eseguire e gli input corrispondenti.

Il file scan.x definisce le strutture dati necessarie, utilizzate come argomento o come valore di ritorno di una funzione remota.

Il Client notifica all'utente l'occorrenza di eventuali errori di comunicazione o di problemi incontrati dal Server durante l'esecuzione delle funzioni.

Il Server invia un codice d'errore nel caso in cui un file o una directory richiesta dal Client non sia presente localmente.

scan.x

```
struct Input_file {
      string fileName <50>;
};

struct Input_dir {
      string dirName <50>;
      int threshold;
};

struct Stat {
      int chars;
      int words;
      int rows;
};

program SCANPROG {
      version SCANVERS {
            Stat FILE_SCAN(Input_file) = 1;
            int DIR_SCAN(Input_dir) = 2;
      } = 1;
} = 0x20000013;
```

Definisce le funzioni remote da implementare e le strutture da loro utilizzate

file_scan - scan_proc.c

```
Stat *file_scan_1_svc(Input_file *argp, struct svc_req *rqstp) {
	static Stat  result;
	FILE *fp;
	char c, prec = EOF;

	result.chars = 0;
	result.words = 0;
	result.rows = 0;

	fp = fopen(argp->fileName, "rt");
	if (fp != NULL) {
		while ((c = fgetc(fp)) != EOF) {
			result.chars += 1;
			if (c == '\n') {
				result.rows += 1;
			}
			if (c == ' ' && prec != '\n' && prec != ' ') {
				result.words += 1;
			}
			prec = c;
		}
		fclose(fp);
	} else {
		result.chars = -1;
		result.words = -1;
		result.rows = -1;
	}

	return &result;
}
```

Inizializza la struttura del valore di ritorno. È valorizzata a -1 in caso di problemi di lettura.

dir_scan - scan_proc.c (1)

```
int *dir_scan_1_svc(Input_dir *argp, struct svc_req *rqstp) {
      static int result;
      DIR *dir;
      struct dirent *ent;
      int fd;
      struct stat path_stat;
      char absFileName[256];
      result = 0;

      if ((dir = opendir(argp->dirName)) != NULL) {
            while ((ent = readdir(dir)) != NULL) {
                  strcpy(absFileName, argp->dirName);
                  strcat(absFileName, "/");
                  strcat(absFileName, ent->d_name);
                  if (ent->d_name[0] == '.') {
                        if (ent->d_name[1] == '.') {
                              if (ent->d_name[2] == '\0') {
                                    continue;
                              }
                        }
                        if (ent->d_name[1] == '\0') {
                              continue;
                        }
                  }

                  stat(ent->d_name, &path_stat);
```

Viene aperto la directory richiesta e controllata la dimensione dei file presenti al suo interno.

dir_scan - scan_proc.c (2)

```
                if (S_ISREG(path_stat.st_mode) == 0) {
                        fd = open(absFileName, O_RDONLY);
                        if (fd != -1) {
                                lseek(fd,0,SEEK_SET);
                                if (lseek(fd, 0, SEEK_END) > (argp->threshold)) {
                                        result++;
                                }
                                close(fd);
                        }
                }
        }
        closedir(dir);
    } else {
        perror("diropen");
        result = -1;
    }

    return &result;
}
```

scan_client.c (1)

```
void scanprog_1(char *host) {
	CLIENT *clnt;
	Stat  *result_1;
	Input_file  file_1_arg;
	int  *result_2;
	Input_dir  dir_1_arg;
	char input[256];

	file_1_arg.fileName = (char*) malloc(DIM);
	dir_1_arg.dirName = (char*) malloc(DIM);

	#ifndef DEBUG
		clnt = clnt_create (host, SCANPROG, SCANVERS, "udp");
		if (clnt == NULL) {
			clnt_pcreateerror (host);
			exit (1);
		}
	#endif  /* DEBUG */

	printf("Inserisci 'f' per la funzione file_scan o 'd' per dir_scan: ");
	while(gets(input)) {
		if (input[0] == 'f') {
			printf("Inserisci il nome di un file remoto: ");
			gets(file_1_arg.fileName);
			result_1 = file_scan_1(&file_1_arg, clnt);
			if (result_1 == (Stat *) NULL) {
				clnt_perror(clnt, "call failed");
			}
```

Richiede l'input all'utente, stampa i risultati e controlla la ricezione di eventuali errori.

scan_client.c (2)

```
            if (result_1->chars == -1) {
                  printf ("Errore di lettura del file.\n");
            } else {
                  printf("File %s:\n\tcaratteri:%d\n\tparole:%d\n\trighe:%d\n", file_1_arg.fileName, result_1->chars, result_1->words, result_1->rows);
            }
      } else if (input[0] == 'd') {
            printf("Inserisci il nome di una directory remota: ");
            gets(dir_1_arg.dirName);
            printf("Inserisci un intero (num. di byte minimi): ");
            scanf("%d", &(dir_1_arg.threshold));
            getchar();
            result_2 = dir_scan_1(&dir_1_arg, clnt);

            if (result_2 == (int *) NULL) {
                  clnt_perror(clnt, "call failed");
            }
            if (*result_2 == -1) {
                  printf("Errore di lettura della directory.\n");
            } else {
                  printf("Directory %s:\n\tFile con dimensione>%d: %d\n", dir_1_arg.dirName, dir_1_arg.threshold, *result_2);
            }
      }

      printf("Operazione terminata.\n- - - - - -\n");
      printf("Inserisci 'f' per la funzione file_scan o 'd' per dir_scan: ");
   }

   #ifndef DEBUG
      clnt_destroy (clnt);
   #endif
}
```