ESERCITAZIONE 9

RPC – Servizio di gestione della votazione

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

Introduzione

Il Client esegue una prima verifica dei parametri ottenuti tramite stdin per quel che riguarda l'operazione da eseguire:

- C = visualizza classifica;
- V = esprimi voto:
  - A = aggiungere un voto;
  - S = sottrarre un voto.

Il Server invece genera una tabella, come da specifica, con valori parzialmente casuali e implementa le due funzioni:

`Output *classifica_giudici_1_svc(void *in, struct svc_req *rqstp)`: per ogni giudice viene calcolato il punteggio in base ai partecipanti e successivamente viene ordinato il risultato, restituendo al Client l'array ordinato di Giudici.

`int *esprimi_voto_1_svc(Input *input, struct svc_req *rqstp)`: si verifica l'esistenza del partecipante passato dal Client e si aggiunge o sottrae un voto, restituendo al Client un valore diverso da -1.

Algoritmo Client

```
input.nomeCandidato = (char*) malloc(128);

printf("Inserire:\nC) per visualizzare la classifica dei giudici\nV) per esprimere un voto\n^D per terminare: ");

while (gets(azione)) {
      if (strcmp(azione, "V") == 0) {
            printf("Inserisci il nome del partecipante \n");
            scanf("%s", input.nomeCandidato);
            getchar();
            do {
                  printf("Aggiungi voto (A), sottrai voto (S):\n");
                  ch = getchar();
                  getchar();
                  if (ch == 'A' || ch == ’S’) {
                        input.tipoOp = ch;
                  }
            } while (ch != 'A' && ch != 'S');

            ris = esprimi_voto_1(&input, cl);
            if (ris == NULL) {
                  clnt_perror(cl, host);
                  exit(1);
            }

            if (*ris < 0) {
                  printf("Errore nell'attribuzione del voto: partecipante non trovato\n");
            } else {
                  printf("Votazione effettuata!\n\n");
            }
      }
```

Algoritmo Server – classifica_giudici (1)

```
Output * classifica_giudici_1_svc(void *in, struct svc_req *rqstp) {
	inizializza();
	int i, j;
	int index = 0, max = 0, now = 0, getG, gDiff, strcmpPlaceholder = -1;
	static Output localOut;

	for (i = 0; i < NUMGIUDICI; i++) {
		localOut.giudici[i].nome = (char*) malloc(MAXSTRINGLENGHT);
		output.giudici[i].punteggioTot = 0;
	}

	printf("\nRicevuta richiesta di stampa della classifica dei giudici.\n");

	for (i = 0; i < NUMGIUDICI; i++) {
		for (j = 0; j < NUMPART; j++) {
			if (strcmp(output.giudici[i].nome, tabella.persona[j].giudice) == 0) {
				output.giudici[i].punteggioTot += tabella.persona[j].voto;
			}
		}
	}

	for (i = 0; i < NUMGIUDICI; i++) {
		if (output.giudici[i].punteggioTot > max) {
			max = output.giudici[i].punteggioTot;
			getG = i;
		}
	}

	localOut.giudici[0] = output.giudici[getG];
```

Si assegna ad ogni giudice i voti corrispettivi di ogni partecipante e si cerca il giudice con il punteggio più alto, ponendolo primo nella lista di output.

Algoritmo Server – classifica_giudici (2)

```
while (index != NUMGIUDICI - 1) {
      for (i = 0; i < NUMGIUDICI; i++) {
            if (strcmpPlaceholder == -1) {
                  if (i == getG) {
                        continue;
                  }
                  if (output.giudici[i].punteggioTot < max && output.giudici[i].punteggioTot > now) {
                        getG = i;
                        now = output.giudici[i].punteggioTot;
                  }
                  if (getG != i && output.giudici[i].punteggioTot == output.giudici[getG].punteggioTot) {
                        if ((gDiff = strcmp(output.giudici[getG].nome, output.giudici[i].nome)) != 0) {
                              if (gDiff > 0) {
                                    getG = i;
                              } else {
                                    strcmpPlaceholder = i;
                              }
                        }
                  }
            } else {
                  getG = strcmpPlaceholder;
                  strcmpPlaceholder = -1;
                  break;
            }
      }
      now = 0;
      max = output.giudici[getG].punteggioTot;
      index++;
      localOut.giudici[index] = output.giudici[getG];
}

return (&localOut);
}
```

Dopo aver trovato il valore massimo, l'algoritmo di sorting si occupa di ordinare gli altri valori e verifica anche i casi di punteggio medesimo.

Algoritmo Server – esprimi_voto

```c
int * esprimi_voto_1_svc(Input *input, struct svc_req *rqstp) {
	inizializza();
	static int res = -1;
	int i, found = -1;
	printf("Ricevuta richiesta di votazione.\n");

	for (i = 0; i < NUMPART && found == -1; i++) {
		if (strcmp(input->nomeCandidato, tabella.persona[i].candidato) == 0) {
			found = i;
		}
	}

	if (found > -1) {
		if (input->tipoOp == 'A') {
			tabella.persona[found].voto++;
			printf("Voto aggiunto a %s, con un totale attuale di %d punti!\n", tabella.persona[found].candidato, tabella.persona[found].voto);
		} else {
			if (tabella.persona[found].voto > 0) {
				tabella.persona[found].voto--;
				printf("Voto tolto a %s, con un totale attuale di %d punti.\n", tabella.persona[found].candidato, tabella.persona[found].voto);
			}
		}
		res = found;
	}

	return(&res);
}
```

Si controlla l'esistenza del partecipante passato dal Client tramite strcmp; se trovato, si aggiunge il voto nel caso in cui tipoOp == 'A' e lo si toglie in tutti gli altri casi, dato che il Client controlla che passino solo 'A' e 'S'.

Conclusione

Durante lo svolgimento del progetto non sono stati riscontrati problemi con l'utilizzo del RPC, ma sono stati fatti perlopiù errori di sintassi che hanno rallentato il progetto e hanno tolto tempo allo sviluppo dell'algoritmo di sorting.

Quest'ultimo infatti non funziona correttamente; ad un primo test effettuato risultava funzionante, ma in realtà è stato solo un caso fortunato, infatti è stato determinato che funziona solo nei casi in cui:

- tutti i valori sono diversi tra loro;
- solo gli ultimi due valori sono identici.