Relatório de FSO - Programming Assignment 1

O objetivo do trabalho prático foi completar um calendário em C. Adicionou-se mais comandos e funcionalidades, o calendário permite agora remover e listar um evento, permite a conexão de vários clientes e notifica ainda os clientes no inicio e durante o respectivo evento.

Step 1

Numa fase inicial foi adicionado o comando remove e “list" que permitem ao cliente remover um evento adicionado ou listar o mesmo:

```
    if (calendar_remove(event_op.id) == -1)
                        reply = -1;
                else
                        reply = event_op.id;

            myWriteSocket(socket, (char *) &reply, sizeof(int));
```

Este comando vai verificar se o evento com o id event_op.id existe no calendário e depois passar o resultado da procura para a classe server com a função myWriteSocket. Caso o resultado da procura seja igual -1, o evento não vai ser removido. Caso o resultado da procura seja diferente de -1 e maior que 0, então o evento com o id event_op.id vai ser removido.

```
EventList* eventList = calendar_list_event(event_op.id);
            if (eventList == NULL) {
                  int reply = -1;
                  myWriteSocket(socket, (char *) &reply, sizeof(int));
            }
            else
            {
                  myWriteSocket(socket, (char *) &eventList->nEvents,
sizeof(int));
                  if (eventList->nEvents)
                        myWriteSocket(socket, (char *) eventList->events,
sizeof(Event) * eventList->nEvents);
                  destroy_event_list(eventList);
            }
```

Este comando vai listar as informações sobre evento com o id event_op.id. Depois de tentar guardar a informação do evento, vai-se verificar se foi guardado alguma informação. Caso não se tenha guardado nada, o evento não poderá ser listado, pois este não existe. Caso se tenha guardado alguma coisa, vai-se listar toda a informação sobre o evento.
Depois de ser listado, vai-se eliminar o apontador onde a informação ficou guardado a informação do evento.

Step 2

Na segunda fase o programa foi melhorado e o servidor pode agora lidar com mais que um cliente. Na fase inicial o servidor era obrigado a esperar que o primeiro cliente terminasse a sua execução para depois dar inicio à execução do próximo cliente. Utilizando o sistema multi-thread o servidor cria um thread para cada cliente que se conectar ao servidor. Alterou-se o return da função “handleClient()” de void para void* para que esta função possa ser chamada sempre que é criado um novo cliente.
Para não haver nenhuma perca de informação foi necessário bloquear as partes criticas do código através de mutex’s não permitindo assim que dois threads alterem ao mesmo tempo certas variáveis que armazenam informação importante.

Step 3

Numa ultima fase do projeto foi adicionada a funcionalidade de notificação. O programa é agora capaz de notificar o cliente quando o evento começa e enquanto este está a decorrer.
Para isso foi adicionado uma nova função:

```
void* notification(void* x) {
    Request* request = (Request*) x;

    while(1){
      pthread_mutex_lock(&lock);
      operation(request) = LIST_ONGOING_EVENTS;
      communicate_event_request(request);
      sleep(60);
      pthread_mutex_unlock(&lock);
    }
}
```

Sempre que é criado um cliente são criados dois threads, um que lida com as notificações e outro que lida com a consola. O método notification(void* x) é utilizado pelo thread que lida com as notificações, este thread entra num loop infinito para estar constantemente a verificar se existem eventos a decorrer. Para dar oportunidade ao thread que lida com a consola de continuar a sua execução, o thread que lida com as notificações é posto em “sleep” e a cada 60 segundos verifica se há alguma notificação.

Execução teste

```
Last login: Mon Nov 10 15:41:57 on ttys000
10-22-105-233:~ joaoelvas$ cd Documents/fct/fso/p/pa1
10-22-105-233:pa1 joaoelvas$ ./client test
> add test 10 11 2014 1545 1
Added event number 1
> list 1
Listing 1 events.

 Id: 1
  Date: 10/11/2014  Hour: 15.45
  Duration: 1 minutes
  Description: test
  Created by: test
> listall
Listing 1 events.

 Id: 1
  Date: 10/11/2014  Hour: 15.45
  Duration: 1 minutes
  Description: test
  Created by: test
> add test1 12 12 2014 1212 12
Added event number 2
> remove 2
Removed event number 2
> listall
Listing 1 events.

 Id: 1
  Date: 10/11/2014  Hour: 15.45
  Duration: 1 minutes
  Description: test
  Created by: test
>
******************
*  NOTIFICATION  *
******************
Listing 1 events.

 Id: 1
  Date: 10/11/2014  Hour: 15.45
  Duration: 1 minutes
  Description: test
  Created by: test
```

Conclusão

Após termos terminado o trabalho proposto podemos concluir que a solução apresentada apesar de preencher todos os requisitos não será a mais eficiente. A solução apresentada para as notificações distingue a consola das notificações permitindo que estas corram ao mesmo tempo através do sleep periódico que foi aplicado nas notificações, mas agora suponhamos que ambos os threads decidem escrever ao mesmo tempo no socket, neste caso a solução apresentada iria correr como o previsto mas visto que apenas um pode escrever no socket o segundo iria ter que esperar. Uma solução eficiente seria apenas um escrever no socket. Por erros de compilação não nos foi possível apresentar uma versão mais eficiente.