TÉCNICO LISBOA

Programação de Sistemas Computacionais

LEE

Relatório do Projeto
Sistema de Controlo de Acessos

Duarte Marques - 99139
Mª Francisca Rodrigues - 99156

Grupo 5, L02

2022/2023 – 2º Semestre, P3

Conteúdo

1 Introdução 2

2 Estruturas de dados utilizadas 3

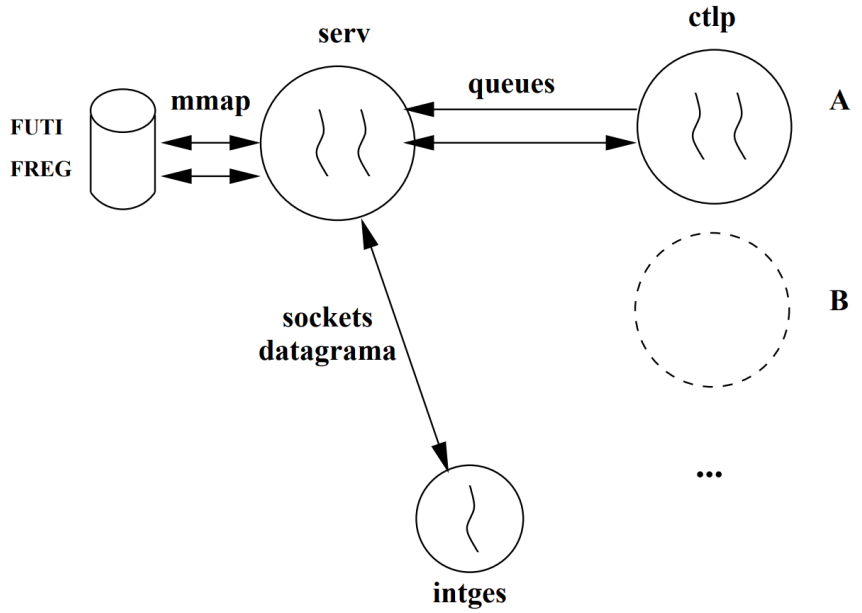
3 Mecanimos de comunicação e sincornização 5

1 Introdução

O principal objetivo deste projeto é permitir que os alunos coloquem em prática os seus conhecimentos adquiridos relativamente aos aspetctos de concorrência, sincronização e comunicação oferecidos por um sistema operativo (Unix/Linux).

Neste trabalho, pretende-se criar uma aplicação (Sistema de Controlo de Acessos) que concretiza a comunicação e sincronização entre o(s) utilizador(es) e um servidor (sistema de controlo), e ainda o registo e consulta de um histórico relativo à evolução do sistema.

Esta aplicação é constituída por três processos fundamentais, conforme ilustrado na imagem seguinte:



1. **intges** - interface para fazer a gestão do sistema
2. **serv** - servidor
3. **ctlp(s)** - o(s) controlador(es) associados às portas

A interface, *intges*, apresenta uma lista de comandos a serem executados, através da sua comunicação com o servidor.
O servidor, *serv*, é o elemento central da aplicação, pois realiza a comunicação com os vários controladores, e ainda, mantém a informação necessária À gestão do sistema.
Por fim, o controlador da(s) porta(s), *ctlp*, está responsavél por controlar o acesso a cada porta. Ao ler o identificador de acesso da porta, valida-o, e dependendo do seu resultado, abre ou não a respetiva porta. Para além disso, vai atualizando o servidor com a informação dos vários acessos concedidos.

2 Estruturas de dados utilizadas

De seguida, são apresentadas as principais estruturas utilizadas neste projeto, bem como uma breve explicação da função de cada uma.

```c
typedef struct cmd_integes {
  char id[10];
  char cmd[10];
  char portas[NPOR+1];
  char estado[3];
  char nome[100];
  char tempo1[100];
  char tempo2[100];
}linha_cmd;

typedef struct uti_s { /* estrutura de um registo utilizador */
  char id[NDIG+1]; /* identificador do utilizador */
  char nome [MAXN]; /* nome do utilizador */
  unsigned char port [NPOR]; /* portas acessíveis ao utilizador: 1-acesso 0-não */
} uti_t;

typedef struct reg_s { /* estrutura de um registo histórico */
  struct timespec t; /* estampilha temporal */
  char p; /* identificador da porta: A,B,C */
  char id [NDIG+1];  /* identificador do utilizador */
} reg_t;

typedef struct  Utilizador {
  char cmd[5];
  char id[3];
  char nome[100];
  char portas[3];
}utilizador;
```

Estas primeiras cinco estruturas estão presentes no ficheiro *ctrl_acess.h*.
A estrutura **linha_cmd** têm como função armazenar várias strings referentes a possíveis argumentos dos vários comandos.
As estruturas **uti_t** e **reg_t**, foram fornecidas pelo docente. A primeira mencionada têm como função o armazenamento de utilizadores do ficheiro *UTILIZADORES.DAT*, e a segunda têm como função armazenar os dados de um registo histórico.
A estrutura **utilizador** têm como função armazenar os caracteres relativos a um utilizador.

```c
/*************************************

Variaveis da queue para mandar comandos
        sobre o CTLP

*************************************/
char queename[10];
int mqids, mqidc, i;
struct mq_attr ma;
struct {
    char    id[TXTSIZE];
    char  porta;
    char    mtext[10];
  } msg;

/*************************************

Variaveis da queue para mandar comandos
        sobre o CTLP

*************************************/
char queename2[10];
int mqids2, mqidc2, j;
struct mq_attr ma2;
struct {
    char    id[TXTSIZE];
    char    mtext[10];
  } msg2;
```

A estrutura msg e msg2 estão contidas no ficheiro *ctlp.c* e no *srv.c*.
A estrutura **msg** envia o id testado na porta para a sua verficação no servidor.
A estrutura **msg2** funciona para a execução dos comandos ao nível do controlador, isto é, o comando *cep* e o *mep*.
Estas estruturas são usadas na comunicação entre o servidor e o(s) controlador(es), tanto para enviar como para receber.

```c
struct  command_integes {
  char* cmd;
  char* argv_integes[4];
  char* help;
  int argc_def;
} const command_integes[] = {
    {"cuti",{"<id>","<nome>","<portas>", "\n"}, "- comando de criar",4},
    {"luti",{"<id>","\n","\n"}, "- comando de listar",2},
    {"euti",{"<id>", "\n", "\n"}, "- comando de eliminar",2},
    {"muti",{"<id>","<portas>", "\n", "\n"}, "- comando de modificar", 3},
    {"lac",{"<portas>", "<id>" ,"<tempo1>", "<tempo2>"}, "- comando de listar",7},
    {"tser",{"\n", "\n", "\n"}, "- comando de terminar",1},
    {"cep",{"<portas>", "\n", "\n"}, "- comando de consultar estado",2},
    {"mep",{"<portas>", "<estado>" ,"\n"}, "- comando de modificar estado",3},
    {"tctl",{"<portas>", "\n", "\n"}, "- comando de controlador",2}
};
```

Por fim, a estrutura **command_integes**, guarda a

3 Mecanimos de comunicação e sincornização

Primeiramente é importante identificar as entidades concorrentes presentes neste trabalho. Estas entidades são os processos e as threads.

Os mecanimos de comunicação utilizados foram: POSIX Message Queues e Sockets Unix Datagram.
As duas queues implementadas neste projeto foram utilizadas para estabelecer a comunicação entre o(s) controlador(es) (ctlp) e o servidor (serv).
O socket datagram foi utilizado na comunicação bidirecional entre a interface de gestão do sistema (intges) e o servidor (serv).

Em termos de sincronização, o mecanismo utilizado entre as várias threads existentes num processo foram os Mutexes.A função dos Mutexes é garantir que o acesso aos ficheiros por parte de processos e threads diferentes, não são executados simultâneamente.