INF1406 Programação Distribuída e Concorrente
2021.2

PUC
RIO

Relatório de testes - Trabalho 2

Professora: Noemi Rodriguez
Grupo: Luiza Del Negro Ciuffo Fernandes - 1721251
Lucas Rebello Damo - 1721275

Teste 1

Parâmetros:

Tipo de cliente: Client 1 uma única requisição por conexão

Tipo de servidor: Servidor 1 um único processo/thread aguarda a conexão e atende todas as requisições do cliente.

Número de rodadas:10.000

Output:

Client 1 - time spent: 2.01240492 sec

Conclusão: (Em considerações finais)

Teste 2

Parâmetros:

Tipo de cliente: Client 1 uma única requisição por conexão

Tipo de servidor: Servidor 2 uma nova conexão para cada requisição

Número de rodadas:10.000

Visualização do código:

Output:

Client 1 - time spent: 2.01509094 sec

Conclusão:(Em considerações finais)

Teste 3

Parâmetros:

Tipo de cliente: Client 1 uma única requisição por conexão

Tipo de servidor: Server 3 servidor concorrente sob demanda de processos

Número de rodadas:10.000

Output:

Client 1 - time spent: 0.00277000 sec

Conclusão:(Em considerações finais)

Teste 4

Parâmetros:

Tipo de cliente: Client 1 uma única requisição por conexão

Tipo de servidor: Server 4 servidor concorrente sob demanda de threads

Número de rodadas:10.000

Output: x

Conclusão:(Em considerações finais)

Teste 5

Parâmetros:

Tipo de cliente: Client 2 conexão aberta para todas as requisições

Tipo de servidor: Servidor 1 um único processo/thread aguarda a conexão e atende todas as requisições do cliente.

Número de rodadas:10.000

Output:

Client 2 - time spent: 0.25395301 sec

Conclusão:(Em considerações finais)

Teste 6

Parâmetros:

Tipo de cliente: Client 2 conexão aberta para todas as requisições

Tipo de servidor: Servidor 2 uma nova conexão para cada requisição

Número de rodadas:10.000

Output: -

Conclusão:

Como o Server 2 fecha a conexão a cada requisição, ele fecha a conexão única do Client 1. Dessa forma, para essa combinação não é possível avaliar o desempenho.
(Mais em considerações finais)

Teste 7

Parâmetros:

Tipo de cliente: Client 2 conexão aberta para todas as requisições

Tipo de servidor: Server 3 servidor concorrente sob demanda de processos

Número de rodadas:10.000

Output:

Client 2 - time spent: 2.19467807 sec

Conclusão:(Em considerações finais)

Teste 8

Parâmetros:

Tipo de cliente: Client 2 conexão aberta para todas as requisições

Tipo de servidor: Server 4 servidor concorrente sob demanda de threads

Número de rodadas:10.000

Output: x

Conclusão: (Em considerações finais)

Considerações Finais

De acordo com os testes realizados nos volumes de 10.000 e 100.000 requisições mostrados em resumo nas tabelas abaixo, podemos ver que existem casos a serem analisados.
Para o server(1) com uma conexão que atende todas as requisições, é claro que uma conexão aberta do cliente para todas as requisições seria mais rápido.
No caso do server(2) que abre uma nova conexão para cada requisição, com o client que representa o mesmo fluxo se mostrou possível porém bastante lento. além disso, não foi possível realizar um teste com o client que realiza apenas uma conexão para todas as requisições pois o server a fecha após a primeira requisição atendida.
O server(3) se mostrou muito mais eficiente com um client que realiza uma única requisição por conexão, pois os processos finalizam mais rápido, mostrando que nem sempre vale a pena o client manter sua conexão aberta.
O server 4 não pode ser testado pois a implementação realizada não roda no replit.
Após as tabelas estão exibidos snippets dos códigos utilizados.

10.000 requisições	Client 1- uma única requisição por conexão	Client 2 - conexão aberta para todas as requisições
Server 1 - uma conexão atende todas as requisições	2.01240492 sec	0.25395301 sec
Server 2 - nova conexão para cada requisição	2.01509094 sec	-
Server 3 - concorrente com processos	0.00277000 sec	2.19467807 sec
Server 4 - concorrente com threads	x	x

100.000 requisições	Client 1 - uma única requisição por conexão	Client 2 - conexão aberta para todas as requisições
Server 1 - uma conexão atende todas as requisições	20.59516907 sec	Client 2 - time spent: 2.51085591 sec
Server 2 - nova conexão para cada requisção	20.43149567 sec	-
Server 3 - concorrente com processos	não rodou no replit	não rodou no replit
Server 4 - concorrente com	x	x

threads		

Trechos do código:

Client 1

```
int num_loop =100000;
(...)
 while(num_loop>0){
   if ((rv = getaddrinfo(argv[1], PORT, &hints, &servinfo)) != 0) {
     fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(rv));
     return 1;
   }
   for(p = servinfo; p != NULL; p = p->ai_next) {
     if ((sockfd = socket(p->ai_family, p->ai_socktype,p->ai_protocol)) == -1){
       perror("cliente: socket");
       continue;
     }
     if (connect(sockfd, p->ai_addr, p->ai_addrlen) == -1) {
       close(sockfd);
       perror("cliente: connect");
       continue;
     }
     break;
   }
   if (p == NULL) {
     fprintf(stderr, "cliente: falha em connect\n"); return 2;
   }
   inet_ntop(p->ai_family, get_in_addr((struct sockaddr *)p->ai_addr), s, sizeof
s);
   freeaddrinfo(servinfo);
   int num = rand() % 10;
   char snum[4];
   sprintf(snum, "%d\n", num);
   if (send(sockfd, snum, 1, 0) ==-1){
     perror("send\n");
   }

   if ((numbytes = recv(sockfd, buf, MAXDATASIZE-1, 0)) == -1) {
     perror("recv");
     exit(1);
   }
   buf[numbytes] = '\0';
   close(sockfd);
   num_loop--;
 }
 (...)
```

Client 2

(...)

```
while(num_loop>0){
   //printf("LOOP NUMERO :%d \n",num_loop);
   int num = rand() % 10;
   char snum[4];
   sprintf(snum, "%d", num);
   //printf("Numero gerado: %s\n",snum);
   //printf("cliente 2: conectando a %s\n", s);
   if (send(sockfd, snum, 1, 0) ==-1)//O cliente envia um número de 0 a 9
       perror("send");
   // se a cresposta for 0 eh pq a conexao fechou entao procura outra
   if ((numbytes = recv(sockfd, buf, MAXDATASIZE-1, 0)) == -1) {
     perror("recv");
     exit(1);
   }
  // printf("cliente 2: recebido '%s'\n",buf);
   num_loop--;
 }
```

(...)

Server 1

(...)

```
if (!fork()) {
     close(sockfd);
     while (recv(new_fd, buff, 4, 0)>0){
       if (send(new_fd, "Hello, world!\n", 13, 0) ==-1){
         perror("serv send\n");
       }
     }
     close(new_fd);
     exit(0);
```

(...)

Server 2

(...)

```
 while(1) {
   sin_size = sizeof their_addr;
   new_fd = accept(sockfd, (struct sockaddr *)&their_addr, &sin_size);
   if (new_fd == -1) {
     perror("accept");
     continue;
   }
   inet_ntop(their_addr.ss_family, get_in_addr((struct sockaddr *)&their_addr), s,
sizeof s);
   if (!fork()) {
     close(sockfd);
     recv(new_fd, buff, 4, 0);
     if (send(new_fd, "Hello, world!", 13, 0) ==-1){
       perror("send");
     }
     close(new_fd);
     exit(0);
   }
   close(new_fd);
 }
```

(...)

Server 3

(...)

```
while (recv(new_fd, buff, 4, 0)>0){
     pid = fork();
     if (pid == 0) {//child
       close(sockfd);
       if (send(new_fd, "Hello, world!\n", 13, 0) ==-1){
         perror("serv send\n");
         close(new_fd);
         exit(0);
       }
       close(new_fd);
       exit(0);
     }//child end
     else if (pid < 0){
 /* The fork failed.  Report failure.  */
       perror("serv child error \n");
       close(new_fd);
       exit(0);}
     else{ /* This is the parent process.  Wait for the child  */
       pid = wait(&status);
```

(...)

Server 4

```
void *create_thread(int new_fd, int sockfd){
 close(sockfd);
 if (send(new_fd, "Hello, world!\n", 13, 0) ==-1){
         perror("serv send\n");
         close(new_fd);
         exit(0);
        }
        close(new_fd);
        exit(0);
}
int main(void){
```

(...)

```
   while (recv(new_fd, buff, 4, 0)>0){

     if (pthread_create(&thid, NULL, create_thread(new_fd,sockfd), "thread 1") !=
0) {
        perror("pthread_create() error");
        exit(1);
      }
   }
```

(...)