# MC833 - Tarefa 4

Erik de Godoy Perillo - RA: 135582

28 de abril de 2016

1. Após modificações no código, a saída ficou deste modo:

```
created socket on port #64136, IP 127.0.0.1
```

2. Após modificações no código, a saída ficou deste modo:

```
connected to IP 127.0.0.1, port #64136
```

3. Uma simples chamada a `fork()` pode fazer o trabalho. O processo inicial do servidor, então, serve como um despachante. A cada nova conexão que ele estabelece, um novo processo é criado para ela. No código, checa-se se o pid retornado por fork é zero. Neste caso, o processo foi despachado e então a tarefa da conexão é feita (nesse caso, receber strings do cliente e colocá-las na saída padrão).

4. Devemos lembrar que o valor retornado por `socket()` é um simples *file descriptor*. Assim, ele segue o comportamento do Unix: processos filhos herdam os *file descriptors* do pai, o que são apenas referências aos arquivos em si. Assim, fechar um socket que você não usa em um processo após um `fork()` não interfere no outro processo que o usa e é até necessário para que não fiquem referências ao arquivo em aberto depois dos processos terminarem.

5. –

   (a) Foram usados três clientes.

   Cliente 1:

```
$ ./client localhost
created socket on port #35966, IP 127.0.0.1
oi1
```

   Cliente 2:

```
$ ./client localhost
created socket on port #35968, IP 127.0.0.1
oii2
```

Cliente 3:

```
$ ./client localhost
created socket on port #35970, IP 127.0.0.1
oiii3
```

Sendo as mensagens oi* entradas. Do lado do servidor, após chamá-lo e invocar os clientes mandando mensagens como especificado acima, sua saída ficou assim:

```
$ ./server
[PID 4591] server listening on port #31472
connected to IP 127.0.0.1, port #35966
connected to IP 127.0.0.1, port #35968
connected to IP 127.0.0.1, port #35970
[PID 4593][IP 127.0.0.1, port 32396] oi1
[PID 4595][IP 127.0.0.1, port 32908] oii2
[PID 4597][IP 127.0.0.1, port 33420] oiii3
```

Ao mesmo tempo em que as mensagens eram enviadas (na ordem em que foram aqui mostradas), o comando `tcpdump -i lo` estava em ação:

```
$ sudo tcpdump -i lo
tcpdump:  verbose output suppressed, use -v or -vv for full
protocol decode
listening on lo, link-type EN10MB (Ethernet), capture size
262144 bytes 08:11:01.279942 IP localhost.localdomain.35966
> localhost.localdomain.31472:  Flags [P.], seq
3627854710:3627854715, ack 2242275928, win 342, options
[nop,nop,TS val 18524259 ecr 18521305], length 5
08:11:01.279982 IP localhost.localdomain.31472 >
localhost.localdomain.35966:  Flags [.], ack 5, win 342,
options [nop,nop,TS val 18524259 ecr 18524259], length 0
08:11:03.577935 IP localhost.localdomain.35968
> localhost.localdomain.31472:  Flags [P.], seq
2490601672:2490601678, ack 3107615614, win 342, options
[nop,nop,TS val 18524948 ecr 18521909], length 6
08:11:03.577977 IP localhost.localdomain.31472 >
localhost.localdomain.35968:  Flags [.], ack 6, win 342,
options [nop,nop,TS val 18524948 ecr 18524948], length 0
08:11:06.019491 IP localhost.localdomain.35970
> localhost.localdomain.31472:  Flags [P.], seq
3652956810:3652956817, ack 2238686682, win 342, options
[nop,nop,TS val 18525681 ecr 18522355], length 7
08:11:06.019582 IP localhost.localdomain.31472 >
localhost.localdomain.35970:  Flags [.], ack 7, win 342,
options [nop,nop,TS val 18525681 ecr 18525681], length 0
```

Como se nota, os pacotes TCP foram de fato enviados dos clientes ao servidor. O pacote do tempo 08:11:01.279982 serve ao cliente 1, o do tempo 08:11:03.577977 ao cliente 2 e o do tempo 08:11:06.019582 ao cliente 3 (todos são acks).

(b) Foi modificado o programa `server.c` de tal modo que o PID do processo inicial e o PID dos filhos fosse mostrado. No exemplo abaixo, três clientes se conectaram ao servidor.

```
$ ./server
[PID 24167] server listening on port #31472
connected to IP 127.0.0.1, port #22666
connected to IP 127.0.0.1, port #23178
connected to IP 127.0.0.1, port #23690
[PID 24173][IP 127.0.0.1, port 23690] oi
[PID 24171][IP 127.0.0.1, port 23178] e ai glr
[PID 24169][IP 127.0.0.1, port 22666] opaa
```

Então temos quatro PIDs. Para checar se os três últimos são realmente filhos do primeiro, podemos usar a ferramenta `ps` para isso. A *flag* `-ppid` lista os filhos do PID especificado:

```
$ ps -ppid 24167 -o pid
24169
24171
24173
```

6. Após o final da conexão, o lado do cliente fica no estado `TIME_WAIT`. Isso condiz, pois implementamos um servidor concorrente que está a espera de outras conexões. A conexão TCP é encerrada, o cliente fica em `TIME_WAIT` e o servidor fica em `LISTEN` como sempre.

   A ferramenta `netstat` pode ser usada para averiguar isso. O servidor foi executado (na porta 31472) e o cliente abriu uma conexão (na porta 36240). Após terminar o cliente, usou-se a ferramenta:

   ```
   $ sudo netstat -ap | grep ":36240|:31472"
   tcp 0 0 *:31472 *:* LISTEN 5614/./server
   tcp 0 0 localhost.localdo:36240 localhost.localdo:31472
   TIME_WAIT -
   ```

   E de fato: a conexão referente ao cliente (pela porta 36240) está em estado `TIME_WAIT`.