



Sistemas Operacionais

Atividade Prática sobre Processos

Aluno: Richard Ferreira Salviano

Matrícula: 202011250024

OBS: Cada questão vale 1,0.

Neste roteiro iremos exercitar alguns conceitos estudados em sala de aula sobre processos e *threads*. Siga os passos abaixo atentando para as questões que precisam ser respondidas (**destacadas em amarelo**). Tente responder por conta própria, seu aprendizado só irá ocorrer assim!

Para esta prática você irá precisar de um ambiente Linux para executar o código, uma vez que faremos uso do padrão POSIX.

Algumas orientações:

- Elabore um documento (.doc, .docx, etc.) contendo as respostas para as questões e os PRINTS das telas;
- Ao elaborar o relatório desse roteiro, FAÇA PRINTS DA TELA SEMPRE QUE POSSÍVEL.

Roteiro

1. Faça *download* do [arquivo](#) para a prática
2. Abra um terminal
3. Descompacte o arquivo. No terminal digite (você precisa estar na pasta onde

foi realizado o *download* do arquivo):

tar xvzf process_exercise.tar.gz

```
rick@PC-Rick: ~/Documentos/SO
rick@PC-Rick:~/Documentos/SO$ tar xvzf process_exercise.tar.gz
processdemo.c
simple_fork.c
simple_fork_inf_loop.c
simple_for_with_pid.c
threaddemo.c
```

4. Encontre o arquivo **processdemo.c** dentro da pasta descompactada. O programa usa a chamada **fork()** para criar um processo filho. Os processos pai e filho são executados separadamente e cada um chama a função **adjustX()** com parâmetros diferentes em cada processo
5. Examine o código-fonte e tente determinar o que ele faz. Você pode fazer isso usando o programa **gedit**. Neste caso, você poderia digitar no terminal: **gedit processdemo.c**

```
4 #include <stdio.h>
5 #include <stdlib.h>
6
7 const clock_t MAXDELAY = 2000000;
8 int x = 50; /* a global variable */
9
10 void delay(clock_t ticks) { /* a "busy" delay */
11     clock_t start = clock();
12     do
13         ; while (clock() < start + ticks);
14 }
15
16 void adjustX(char * legend, int i) {
17     while (1) /* loop forever */
18     {
19         printf("%s: %i\n", legend, x);
20         x += i;
21         delay(rand()%MAXDELAY);
22     }
23 }
24
25 main()
26 {
27     int c;
28     srand(time(NULL));
29     printf("creating new process:\n");
30     c = fork();
31     printf("process %i created\n", c);
32     if (c==0)
33         adjustX("child", 1); /* child process */
34     else
35         adjustX("parent", -1); /* parent process */
36 }
```

6. Faça um exercício de tentar prever o que o código vai fazer
7. Compile o código-fonte. No terminal digite:

gcc processdemo.c -o processdemo

```
rick@PC-Rick:~/Documentos/SO$ gcc processdemo.c -o processdemo
processdemo.c: In function 'delay':
processdemo.c:11:19: warning: implicit declaration of function 'clock' [-Wimplicit-function-declaration]
   11 |     clock_t start = clock();
       |                      ^~~~~
processdemo.c: At top level:
processdemo.c:24:1: warning: return type defaults to 'int' [-Wimplicit-int]
   24 | main()
       | ^~~~~
processdemo.c: In function 'main':
processdemo.c:26:10: warning: implicit declaration of function 'time' [-Wimplicit-function-declaration]
   26 |     srand(time(NULL));
       |           ^~~~~
processdemo.c:28:8: warning: implicit declaration of function 'fork' [-Wimplicit-function-declaration]
   28 |     c = fork();
       |         ^~~~~
```

8. Rode o programa. No terminal digite:

./processdemo

```
rick@PC-Rick:~/Documentos/SO$ ./processdemo
creating new process:
process 6891 created
parent: 50
process 0 created
child: 50
parent: 49
child: 51
parent: 48
child: 52
parent: 47
child: 53
parent: 46
child: 54
parent: 45
child: 55
```

9. Descreva a saída e explique por que ela é dessa forma.

Estão sendo criados processos e em cada linha está saindo um processo pai e um filho separadamente. A partir de forks estão sendo criados os processos filhos. Em cada processo têm parâmetros diferentes e são imprimidos a legenda dele e um contador indicando o número do processo, o processo pai diminuindo 1 e o processo filho aumentando 1, tudo isso dentro de um loop infinito.

10. Enquanto o programa roda, execute (em outro terminal) o seguinte comando para visualizar os processos existentes na sua máquina:

ps xl

```
rick@PC-Rick: ~/Documentos/So$ ps xl
 4 1000 4760 4743 20 0 33860152 59960 do_wai S ? 0:00 /opt/google/chrome-unstable/chrome --type=zygote --enable-crashpad --crashp
4 1000 4761 4760 20 0 33563784 4324 skb_wa S ? 0:00 /opt/google/chrome-unstable/nacl_helper
5 1000 4764 4760 20 0 33860180 16544 do_pol S ? 0:00 /opt/google/chrome-unstable/chrome --type=zygote --enable-crashpad --crashp
1 1000 4786 4759 20 0 34180932 187076 do_pol SL ? 1:08 /opt/google/chrome-unstable/chrome --type=gpu-process --enable-crashpad --c
0 1000 4792 4743 20 0 33934060 113376 futex_ SL ? 0:15 /opt/google/chrome-unstable/chrome --type=utility --utility-sub-type=networ
1 1000 4808 4764 20 0 33909756 47672 futex_ SL ? 0:00 /opt/google/chrome-unstable/chrome --type=utility --utility-sub-type=storag
1 1000 5240 4764 20 0 1184769560 95156 futex_ SL ? 0:00 /opt/google/chrome-unstable/chrome --type=renderer --enable-crashpad --cras
1 1000 5302 4764 20 0 1184789160 121260 futex_ SL ? 0:05 /opt/google/chrome-unstable/chrome --type=renderer --enable-crashpad --cras
1 1000 5346 4764 20 0 1184777756 100444 futex_ SL ? 0:00 /opt/google/chrome-unstable/chrome --type=renderer --enable-crashpad --cras
0 1000 5375 4743 20 0 207124 33232 do_pol SL ? 0:00 /usr/bin/python3 /usr/bin/chrome-gnome-shell chrome-extension://gphhapmejob
0 1000 5519 4743 20 0 34171588 72848 futex_ SL ? 0:00 /opt/google/chrome-unstable/chrome --type=utility --utility-sub-type=audio.
1 1000 5558 4764 20 0 1185861836 248304 futex_ SL ? 0:26 /opt/google/chrome-unstable/chrome --type=renderer --enable-crashpad --cras
1 1000 5572 4764 20 0 1184825276 263792 futex_ SL ? 0:12 /opt/google/chrome-unstable/chrome --type=renderer --enable-crashpad --cras
1 1000 5588 4764 20 0 1185858596 173780 futex_ SL ? 0:01 /opt/google/chrome-unstable/chrome --type=renderer --enable-crashpad --cras
4 1000 5654 3029 20 0 38301172 157400 do_pol SL ? 0:15 /snap/discord/143/usr/share/discord/Discord --use-tray-icon --no-sandbox --
0 1000 5743 5654 20 0 214752 48196 do_pol S ? 0:00 /snap/discord/143/usr/share/discord/Discord --type=zygote --no-zygote-sandb
0 1000 5744 5654 20 0 214756 47944 do_pol S ? 0:00 /snap/discord/143/usr/share/discord/Discord --type=zygote --disable-seccomp
1 1000 5770 5743 20 0 513352 132296 do_pol SL ? 0:09 /snap/discord/143/usr/share/discord/Discord --type=gpu-process --field-tria
0 1000 5781 5654 20 0 276904 73884 futex_ SL ? 0:01 /snap/discord/143/usr/share/discord/Discord --type=utility --utility-sub-ty
0 1000 5821 5654 20 0 80663988 360820 futex_ SL ? 1:27 /snap/discord/143/usr/share/discord/Discord --type=renderer --disable-secco
0 1000 5851 5654 20 0 546664 55936 futex_ SL ? 0:00 /snap/discord/143/usr/share/discord/Discord --type=utility --utility-sub-ty
0 1000 6085 3003 20 0 500668 31056 do_pol SL ? 0:00 update-notifier
1 1000 6252 4764 20 0 1184781000 159896 futex_ SL ? 0:15 /opt/google/chrome-unstable/chrome --type=renderer --enable-crashpad --cras
1 1000 6277 4764 20 0 1185890128 347936 futex_ SL ? 2:20 /opt/google/chrome-unstable/chrome --type=renderer --enable-crashpad --cras
0 1000 6346 4764 20 0 1184769708 120388 futex_ SL ? 0:01 /opt/google/chrome-unstable/chrome --type=renderer --enable-crashpad --cras
0 1000 6470 2752 20 0 324136 8952 do_pol SL ? 0:00 /usr/libexec/gvfsd-trash --spawner :1.2 /org/gtk/gvfs/exec_spaw0
0 1000 6493 2752 20 0 398016 9348 do_pol SL ? 0:00 /usr/libexec/gvfsd-network --spawner :1.2 /org/gtk/gvfs/exec_spaw1
0 1000 6506 2752 20 0 325812 9148 do_pol SL ? 0:00 /usr/libexec/gvfsd-dnssd --spawner :1.2 /org/gtk/gvfs/exec_spaw3
0 1000 6577 2722 20 0 830284 60148 do_pol Ssl ? 0:12 /usr/libexec/gnome-terminal-server
0 1000 6596 6577 20 0 23776 9108 do_wai Ss pts/0 0:00 bash
1 1000 7035 4764 20 0 1184771608 96328 futex_ SL ? 0:00 /opt/google/chrome-unstable/chrome --type=renderer --enable-crashpad --cras
1 1000 8483 4764 20 0 1184744968 65496 futex_ SL ? 0:00 /opt/google/chrome-unstable/chrome --type=renderer --enable-crashpad --cras
0 1000 8561 6596 20 0 2772 1072 - R+ pts/0 1:25 ./processdeno
1 1000 8562 8561 20 0 2772 96 - R+ pts/0 1:25 ./processdeno
0 1000 8576 2722 20 0 1285752 76828 do_pol SL ? 0:01 /usr/bin/nautilus --gapplication-service
0 1000 8602 6577 20 0 23776 9212 do_wai Ss pts/1 0:00 bash
4 1000 8774 8602 20 0 21408 3508 - R+ pts/1 0:00 ps xl
```

11. Qual o processo pai e qual o processo filho? (Dica, verifique a coluna PID e PPID. Se não souber o que é PID e PPID, procure no Google). Justifique sua resposta.

O PID é o número de identificação do processo ativo que no caso do pai é 8561 e o filho é o 8562, isso é garantido pelo fato que no PPID, que no caso é o número de identificação do processo-pai, no processo 8562 ele tem como pai justamente o 8561.

12. Use o comando "kill -9 <PID do filho>" para matar o processo filho. O que aconteceu?

```
0  1000   8561   6596  20   0   2772  1072  -   R+  pts/0   15:17  ./processdemo
1  1000   8562   8561  20   0     0     0  -   Z+  pts/0   15:14  [processdemo] <defunct>
```

Aconteceu que o filho foi morto e isso é comprovado pois no fim do processo ele aparece com um "defunct", que seria algo como defunto/extinto/morto.

```
child: 968
parent: -869
child: 969
parent: -870
parent: -871
parent: -872
parent: -873
parent: -874
parent: -875
parent: -876
parent: -877
```

E começou a rodar imprimindo só os pais no outro terminal.

13. Use o comando "kill -9 <PID do pai>" para matar o processo pai. O que aconteceu?

```
Morto
rick@PC-Rick:~/Documentos/SO$
```

Onde estava rodando foi declarado como morto a partir dessa mensagem e no ps xl não aparece mais os processos.

14. Rode o programa novamente. Identifique e mate o processo pai primeiro e em seguida o filho. O que aconteceu?

```
parent: -52
child: 152
Morto
rick@PC-Rick:~/Documentos/SO$ child: 153
child: 154
child: 155
```

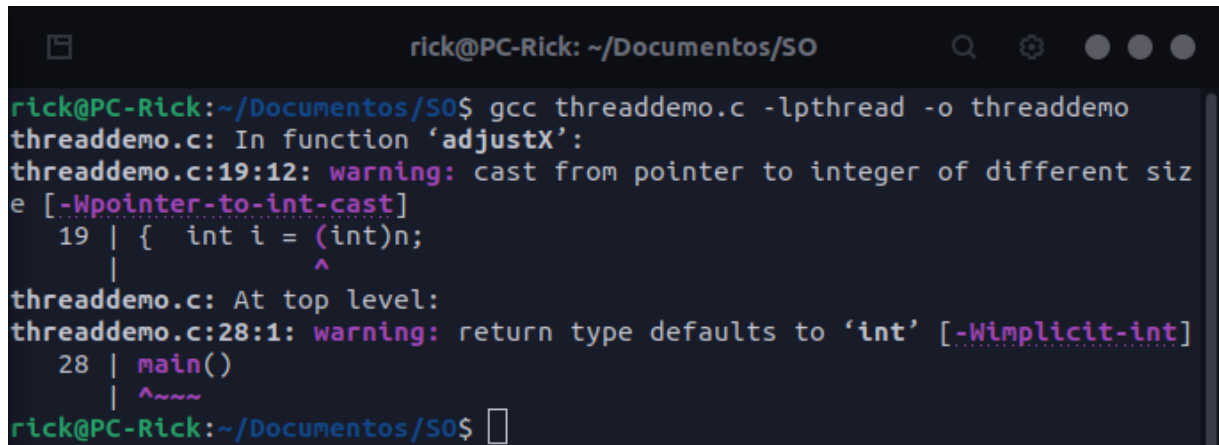
O processo filho continua rodando e o pai é morto e só depois que se mata o filho ele para

15. Faz diferença matar o pai ou o filho antes?

Aparentemente não faz nenhuma diferença, já que observando os dois terminais o mesmo comportamento é percebido.

16. Compile o programa **threaddemo.c** com o comando:

gcc threaddemo.c -lpthread -o threaddemo



```
rick@PC-Rick: ~/Documentos/SO
rick@PC-Rick:~/Documentos/SO$ gcc threaddemo.c -lpthread -o threaddemo
threaddemo.c: In function 'adjustX':
threaddemo.c:19:12: warning: cast from pointer to integer of different size [-Wpointer-to-int-cast]
   19 | {   int i = (int)n;
      |           ^
threaddemo.c: At top level:
threaddemo.c:28:1: warning: return type defaults to 'int' [-Wimplicit-int]
   28 | main()
      | ^~~~
rick@PC-Rick:~/Documentos/SO$
```

17. Esse programa usa a biblioteca *POSIX threads library* e é muito similar a **processdemo.c**, mas usa *threads* em vez de processos

18. Rode o programa. O que ele faz? Qual a diferença dele para o programa **processdemo.c**?

```
rick@PC-Rick:~/Documentos/SO$ ./threaddemo
creating threads:
adjustment = 1; x = 50
adjustment = -1; x = 51
adjustment = 1; x = 50
adjustment = -1; x = 51
adjustment = -1; x = 50
adjustment = 1; x = 49
adjustment = -1; x = 50
adjustment = 1; x = 49
adjustment = 1; x = 50
adjustment = -1; x = 50
adjustment = 1; x = 50
adjustment = 1; x = 51
```

Ao contrário do **processdemo** ele não cria processos e sim threads e cada linha é uma thread. Ele roda dentro de `while true`, ou seja, infinitamente dentro de um loop, imprimindo em cada linha a descrição, o `i` e o `x`, em que o `x` começa com 50 e é somado com o valor de `i` que vem do `pthread_create` e varia em 1 e -1.

19. Qual a diferença de velocidade de saída (medido em linhas por segundo) comparado a **processdemo**? Quem é mais rápido? Você tem uma ideia do porquê?

O **threaddemo** roda bem mais rápido e isso acontece porque threads são mais rápidos de criar e destruir se comparado aos processos, uma vez que não possuem quaisquer recursos associados a eles. O thread é uma forma como um processo/tarefa de um programa de computador é dividido em duas ou mais tarefas que podem ser executadas concorrentemente.

20. Use o comando **ps xl** para verificar que há apenas 1 processo **threaddemo**

```

0 1000 9276 8509 20 0 19164 1168 - Rl+ pts/0 1:05 ./threaddemo
0 1000 9281 2696 20 0 1219756 76936 do_pol Sl ? 0:01 /usr/bin/nautilus --gapplication-service
0 1000 9308 8490 20 0 23776 9184 do_wai Ss pts/1 0:00 bash
4 1000 9480 9308 20 0 21408 3472 - R+ pts/1 0:00 ps xl
rick@PC-Rick:~/Documentos/SO$

```

21. Investigue o efeito de remover o *loop* infinito no fim do **main()**. O que acontece? Por quê?

```

main()
{   int a;
    srand(time(NULL));
    pthread_t  up_thread, dn_thread;

    pthread_attr_t *attr; /* thread attribute variable */
    attr=0;

    printf("creating threads:\n");
    pthread_create(&up_thread,attr, adjustX, (void *)1);
    pthread_create(&dn_thread,attr, adjustX, (void *)-1);
}

```

C ▾ Largura da tabulação: 8 ▾ Lin 38, Col 57 ▾ INS

```

rick@PC-Rick:~/Documentos/SO$ ./threaddemo
creating threads:
adjustment = 1; x = 50
rick@PC-Rick:~/Documentos/SO$

```

O processo nem aparece no `ps xl` e o programa inicia e para no mesmo momento. Pois o programa se inicia imprime o “creating threads” depois ele a partir dos `pthread_create` chama a função `adjustX`, ela roda com os valores iniciais mas como não tem o `while true`, que no caso funciona como um *loop* infinito, já que não tem um `break`, o programa para por ali, até porque a função `main` é onde roda todas as outras funções e nesse caso ela só roda uma vez.

22. Modifique o programa **threaddemo.c** para que a manipulação do contador que é feita pelas diferentes *threads* se assemelhe à manipulação realizada pelos processos em **processdemo.c**


```
threaddemo.c  x  processdemo.c  x
6 #include <stdlib.h>
7 #include <pthread.h>
8
9 const clock_t MAXDELAY = 2000000;
10
11 void delay(clock_t ticks) { /* a "busy" delay */
12     clock_t start = clock();
13     do
14         ; while (clock() < start + ticks);
15 }
16
17 void * adjustX(void *n)
18 {     int i = (int)n;
19     int x = 50;
20     while (1) /* loop forever */
21     {     printf("adjustment = %2i; x = %i\n", i, x);
22         x += i;
23         delay(rand()%MAXDELAY);
24     }
25     return(n);
26 }
27
28 main()
29 {     int a;
30     srand(time(NULL));
31     pthread_t  up_thread, dn_thread;
32
33     pthread_attr_t *attr; /* thread attribute variable */
34     attr=0;
35
36     printf("creating threads:\n");
37     pthread_create(&up_thread,attr, adjustX, (void *)1);
38     pthread_create(&dn_thread,attr, adjustX, (void *)-1);
39
40     while(1){;}
```

A variável que antes estava como global agora é declarada dentro da função `adjustX`. Porque a variável estava sendo compartilhada, então agora ela ficando em um scope local resolve.