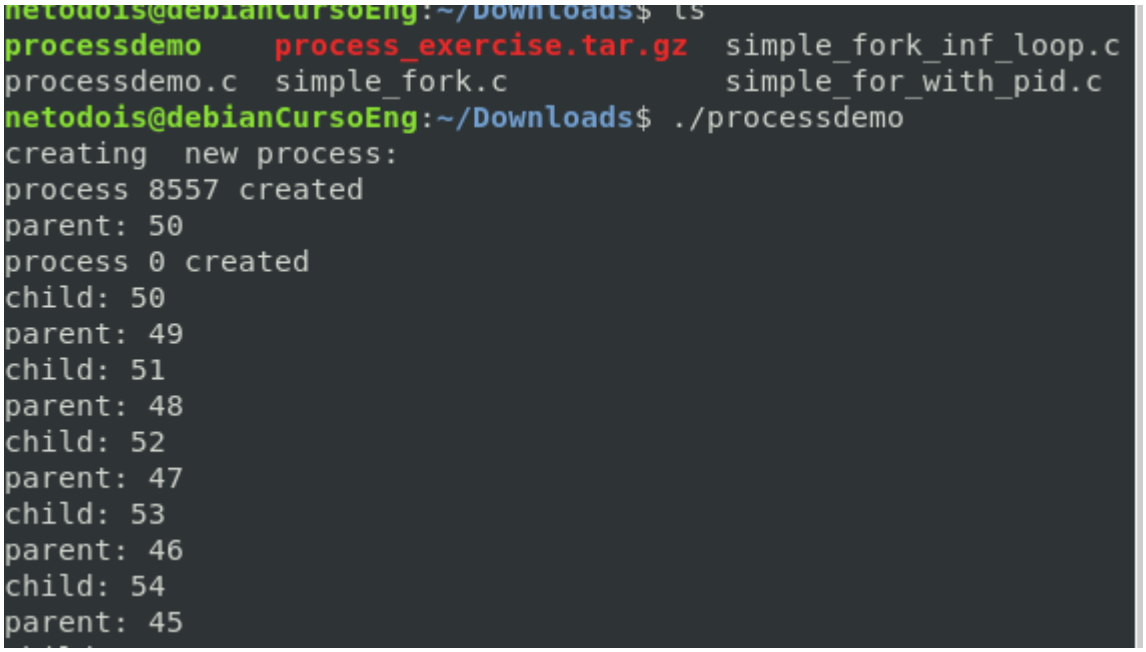
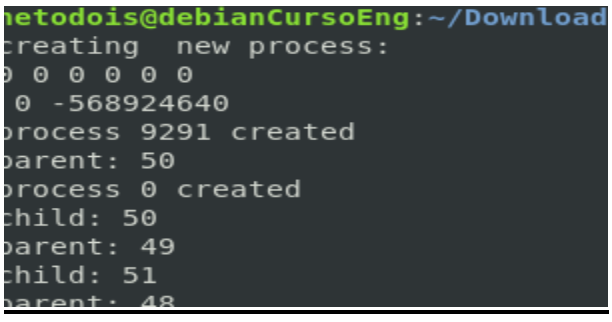
Instituição: IFPB
Curso: Engenharia De Computação
Disciplina: Sistemas Operacionais
Professor: David Candeia Medeiros
Aluno: Luiz Medeiros Neto



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

No programa, são executados dois processos simultaneamente, uma vez que o comando .fork() cria um processo idêntico ao que o chamou. Nesse sentido é possível acompanhar na tela as atualizações das variáveis X(variável global) do processo pai e filho ao mesmo tempo.

A função delay() seta o delay do intervalo de tempo de exibição e incremento da variável X nos dois processos. No processo **PAI o X é incrementado com -1 e no FILHO com 1.**



```
1     0  9258     2  20   0     0     0 -      I    ?
0  1000  9285  9046  20   0  7652  4500 -      Ss   pts/1
1     0  9288     2  20   0     0     0 -      I    ?
0  1000  9290  9145  20   0  2276   676 -      R+   pts/0
1  1000  9291  9290  20   0  2276    80 -      R+   pts/0
0  1000  9293  9285  20   0 10548  1200 -      R+   pts/1
netodois@debianCursoEng:~/Downloads$ sssssssssssssssssssssss
```

Item 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.

Como mostrado acima, assim que o programa entra em execução, é informado o ID(9291) do processo filho, uma vez que ele retorna 0, logo ao observar o PPID na tabela, observa-se que seu pai é o processo 9920.

Itens 12 e 13

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

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

Morte do processo FILHO

```
parent: 2
child: 98
parent: 1
child: 99
parent: 0
parent: -1
parent: -2
parent: -3
parent: -4
```

O processo PAI continua em execução normalmente.

Morte do processo PAI

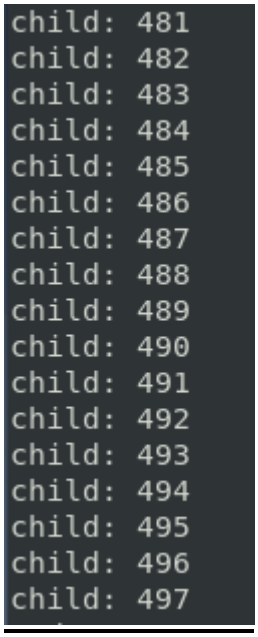
```
parent: -657
parent: -658
parent: -659
parent: -660
parent: -661
parent: -662
Morto
netodois@debianCursoEng:~/Downloads$
```

Após a morte do processo PAI, o programa para e retorna “dead”.

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

Morte do processo PAI

Morte do processo FILHO



Mesmo após a morte do processo “pai”, o processo “filho” continuou em execução, observando-se então um grau de independência. O programa é “re-executado” e processo filho continua em execução e o processo pai foi encerrado.

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

Quando o PAI morre primeiro, o programa é “re-executado”, como mostra a imagem abaixo.

```
child: 178
Morto
netodois@debianCursoEng:~/Downloads$ child: 179
child: 180
child: 181
child: 182
```

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

Enquanto no 1º programa encontravam-se 2 processos, no 2º programa encontra-se apenas 1. Entretanto, são duas threads foram criadas e atuam no mesmo processo no qual sua função é alterar o valor da variável "X" conforme um loop infinito.

```
creating threads:
adjustment = -1; x = 50
adjustment =  1; x = 49
adjustment = -1; x = 50
adjustment =  1; x = 49
adjustment =  1; x = 50
adjustment = -1; x = 51
adjustment =  1; x = 50
adjustment = -1; x = 51
adjustment =  1; x = 50
adjustment = -1; x = 51
adjustment =  1; x = 50
adjustment = -1; x = 51
adjustment =  1; x = 50
adjustment =  1; x = 51
```

Item 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 é mais rápido, uma vez que se trata de threads que por sua vez estão em um mesmo processo. O que destaca também, um dos motivos para o uso de threads, que é o ganho de eficiência por conta do paralelismo de threads.

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

```
threaddemo.c: At top level:
threaddemo.c:28:1: warning: return type defaults to
 main()
 ^~~~
netodois@debianCursoEng:~/Downloads$ ./threaddemo
creating threads:
netodois@debianCursoEng:~/Downloads$
```

As threads não são executadas. Em minha visão, uma vez feita uma análise superficial, supõe-se que as threads precisam estar dentro de um loop infinito para que sejam encerradas conforme a vontade do usuário ou algum erro no Sistema.

Item 22.Modifique o programa threaddemo.c para ele fazer a mesma coisa que processdemo.c

```
metodois@debianCursoEng:~/Downloads$ ./threaddemo
creating threads:
adjustment = -1; x = 50
creating threads:
adjustment =  1; x = 50
adjustment = -1; x = 49
adjustment =  1; x = 51
adjustment = -1; x = 48
adjustment =  1; x = 52
adjustment = -1; x = 47
adjustment =  1; x = 53
^C
metodois@debianCursoEng:~/Downloads$
```

Para que o programa faça a mesma coisa que o **processdemo**, é necessário que seja utilizado o comando **fork() na variável "a" declarada.** Logo, é criado um processo idêntico ao que o chamou e as variáveis X são modificadas separadamente sendo a execução quase idêntica ao primeiro programa.

```
main()
{  int a;
   srand(time(NULL));
   pthread_t  up_thread, dn_thread;
   pthread_attr_t *attr;  /* thread attribute variable */
   attr=0;
   a = fork();
   printf("creating threads:\n");
   a == 0 ? pthread_create(&up_thread,attr, adjustX, (void *)1) :
   pthread_create(&dn_thread,attr, adjustX, (void *)-1);

   while (1) /* loop forever */
   { ;}
```

.