```
/*****************************************************************/
/** MAC 438 - Programação Concorrente                            */
/** IME-USP - Primeiro Semestre de 2016                          */
/** Prof. Marcel Parolin Jackowski                               */
/**                                                              */
/** Primeiro Exercício-Programa                                  */
/** Arquivo: bonus.pdf                                           */
/**                                                              */
/** Wagner Ferreira Alves    7577319                             */
/** Rafael Marinaro Verona   7577323                             */
/** 04/04/2016                                                   */
/*****************************************************************/
```

Neste arquivo, serão analisados os 3 programas bônus do EP1.
Antes de tudo, é importante notar que:

"No UNIX, há somente uma chamada de sistema para criar um novo processo: fork. Essa chamada cria um clone idêntico ao processo que a chamou. Depois da fork, os dois processos, o pai e o filho, têm a mesma imagem de memória, as mesmas variáveis de ambiente e os mesmos arquivos abertos."
Tanenbaum, Sistemas Operacionais Modernos, 3ª edição em português

Ou seja: nos programas seguintes, o fork fará a cópia de toda a memória do processo pai (o que chamou fork inicialmente) para o processo filho e isso será importante para entender o que acontece com o valor do índice i de cada loop.

Quando um processo cria um filho, os filhos podem criar seus processos também, denotando uma árvore de processos.
Ao executar cada programa pela linha de comando, o interpretador de comandos (shell) cria o processo principal com fork.
Segundo a documentação de fork, a funcão retorna:

- 0 ao processo filho criado,
- Pid do processo filho ao pai,
- -1 ao processo pai caso não consiga realizar o fork

Vejamos os detalhes de cada programa a seguir.

Programa 1

```
1  #include <stdio.h>
2  int main(int argc, char **argv) {
3      int i, n = 4;
4      for(i = 1; i < n; i++) {
5          if (fork()) {
6              break;
7          }
8      }
9      printf("Processo %d de pai %d\n", getpid(), getppid());
10     sleep(1);
11     return 0;
12 }
```

Neste programa, valem as observações:

Obs1: em caso de erro do fork na linha 5, o processo pai não cria um descendente e recebe o retorno de fork como -1. Isso faz ele executar a linha 6, saindo do loop e terminando sua execução, fazendo com que o programa não gere mais quaisquer descendentes.

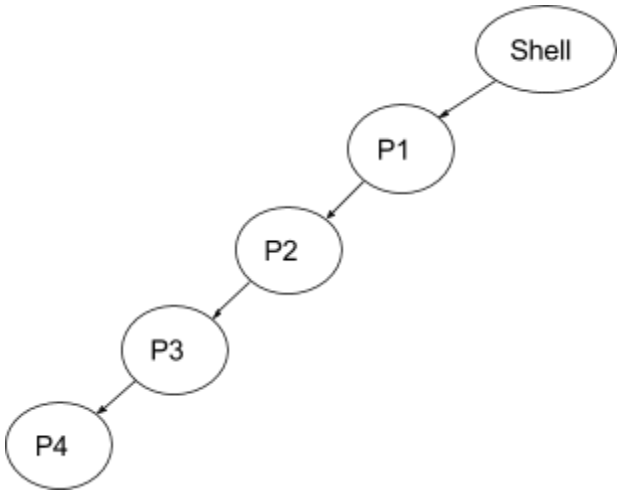
Obs2: em caso de sucesso no fork da linha 5, o processo pai faz a comparação if (pid), onde pid != 0. Isso faz com que ele entre na linha 6, saia do loop e termine na linha 11. Já o processo filho faz a comparação if (0), que é falsa e recebe o valor i atual do pai. Ele então incrementa o i na linha 4 e se i < n, executa a linha 5.

O shell cria o processo P1, que irá executar o programa 1 a partir da linha 1.
fork é chamado na linha 5 e:

- Em caso de erro: vale a Obs1. ***Nesse caso, o programa termina só com P1 criado.***
- Em caso de sucesso: vale a Obs2; P1 termina e P2 é criado com i = 1. P2 executa 5 e:
  - Em caso de erro: vale a Obs1. ***Nesse caso, o programa termina só com P1 e P2 criados.***
  - Em caso de sucesso: vale Obs2; P2 termina e P3 é criado com i = 2. P3 executa 5 e:
    - Em caso de erro: vale Obs1. ***Nesse caso, o programa termina só com P1, P2 e P3 criados.***

- Em caso de sucesso: vale Obs2; P3 termina e P4 é criado com i = 3. P4 executa a linha 4 para fazer i = 4, o que faz ele terminar o loop e P4. ***Nesse caso, o programa termina com P1, P2, P3 e P4 criados.***

Árvore dos processos, em caso de sucesso em todos os forks:



Portanto, são criados **a partir de P1 no máximo 3 outros processos**, como mostra o diagrama acima.

****
*

Programa 2

```
1  #include <stdio.h>
2  int main(int argc, char **argv) {
3      int i, n = 4;
4      for(i = 0; i < n; i++) {
5          if (fork() <= 0) {
6              break;
7          }
8      }
9      printf("Processo %d de pai %d\n", getpid(), getppid());
10     sleep(1);
11     return 0;
12 }
```

Obs1: Na linha 5, percebemos que caso fork retorne -1 ou 0, então a linha 6 será executada, o processo sairá do loop e terminará na linha 11. Ou seja, caso dê erro: a chamada do fork o pai deixa de gerar filhos. Caso estiver sendo executado o código do processo filho: o filho termina.

Nesse caso, o shell cria P1 que, com i = 0 executa a linha 5.

- Se ocorrer um erro: Vale a Obs1. (P1 termina na L11). ***Processos criados: P1.***
- Senão: P1 cria P2 com i = 0 e vale Obs1 para P2 (Termina na L11).

P1 faz i = 1 e executa a linha 5.

- Se ocorrer um erro: Vale a Obs1. (P1 termina na L11). ***Processos criados: P1, P2.***
- Senão: P1 cria P3 com i = 1 e vale Obs1 para P3 (Termina na L11).

P1 faz i = 2 e executa a linha 5.

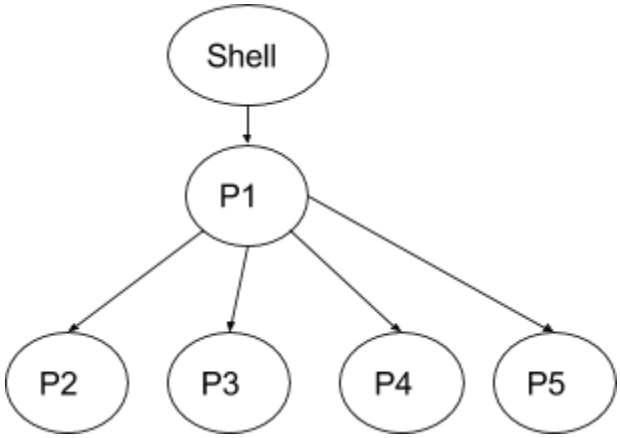
- Se ocorrer um erro: Vale Obs1. (P1 termina na L11). ***Processos criados: P1, P2, P3.***
- Senão: P1 cria P4 com i = 2 e vale Obs1 para P4 (Termina na L11).

P1 faz i = 3 e executa a linha 5

- Se ocorrer um erro: Vale Obs1. (P1 termina na L11). ***Processos criados: P1, P2, P3, P4.***
- Senão: P1 cria P5 com i = 3 e vale Obs1 para P5 (Termina na L11).

P1 faz i = 4 e não executa mais o loop; então, P1 termina na L11. ***Processos criados: P1, P2, P3, P4, P5.***

Árvore dos processos, em caso de sucesso em todos os forks:



Portanto, **a partir de P1 são criados no máximo 4 processos novos.**

****
*

Programa 3

```
1  #include <stdio.h>
2  int main(int argc, char **argv) {
3      int i, n = 3;
4      for(i = 0; i < n; i++) {
5          if (fork() == -1) {
6              break;
7          }
8      }
9      printf("Processo %d de pai %d\n", getpid(), getppid());
10     sleep(1);
11     return 0;
12 }
```

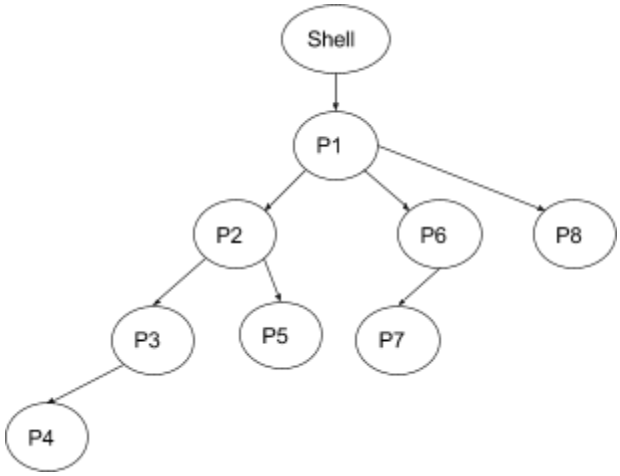
Nesse programa, vale que se o fork retornar erro (-1), então o processo que o chamou sai do loop e termina sem gerar mais descendentes, porém os processos anteriores continuam executando e podem talvez gerar novos descendentes.

Para simplificar, vamos ver o que acontece caso fork sempre retorne sucesso:

- O shell cria P1
- P1 executa até a linha 4, recebendo i1 = 0.
- Na linha 5, P1 chama o fork e cria P2, que recebe i2 = 0.

- P2 executa a linha 4 fazendo i2 = 1.
- P2 executa a linha 5 e cria P3, com i3 = 1.
- P3 executa a linha 4 e faz i3 = 2.
- P3 executa a linha 5 e cria P4, com i4 = 2.
- P4 executa a linha 4 e faz i4 = 3.
- Como a comparação 3 < 3 falha, P4 termina sem gerar novos descendentes.
- P3 agora executa a linha 4 e faz i3 = 3.
- Como a comparação 3 < 3 falha, P3 termina sem gerar novos descendentes.
- P2 agora executa a linha 4 e faz i2 = 2.
- P2 executa a linha 5 e cria P5 com i5 = 2.
- P5 executa a linha 4 e faz i5 = 3.
- Como a comparação 3 < 3 falha, P5 termina sem gerar novos descendentes.
- P1 agora executa a linha 4 e faz i1 = 1.
- P1 executa a linha 5 e cria P6, com i6 = 1.
- P6 executa a linha 4 e faz i6 = 2.
- P6 executa a linha 5 e cria P7 com i7 = 2.
- P7 executa a linha 4 e faz i7 = 3.
- Como a comparação 3 < 3 falha, P7 termina sem gerar novos descendentes.
- P6 agora executa a linha 4 e faz i6 = 3.
- Como a comparação 3 < 3 falha, P6 termina sem gerar novos descendentes.
- Agora P1 executa a linha 4 e faz i1 = 2.
- P1 executa a linha 5 e cria P8, com i8 = 2.
- P8 executa a linha 4 e faz i8 = 3.
- Como a comparação 3 < 3 falha, P8 termina sem gerar novos descendentes.
- P1 agora executa a linha 4 e faz i1 = 3.
- Como a comparação 3 < 3 falha, P1 termina sem gerar novos descendentes.

Árvore dos processos, em caso de sucesso em todos os forks:



Portanto, **a partir de P1 são criados no máximo 7 processos novos.**