# Lab02 – Processos

**(★★★)** Escreva o programa “arvore-processos.c” que executa o programa mostrado na Tabela 1. O programa cria uma estrutura de árvore de processos do tipo P_A -> P_B -> P_C, onde o processo P_A é o pai de P_B, quem por sua vez é o pai de P_C. A sua solução deverá incluir um diagrama de processos e chamadas de sistema similar ao da Figura 3.9 do livro-texto:

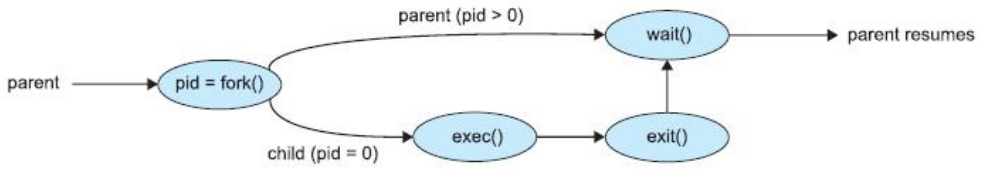


**Figure 3.9** Process creation using the `fork()` system call.

N.B.: O PID é um identificador de valor único que o SO atribui a cada processo. Após a criação de um processo filho, o “PID interno” é o identificador do processo que recebe o processo pai dentro do seu corpo de programa.

- **Sugestões:**
  - Revise o código da Figura 3.8 do livro-texto
  - Revise o manual man das seguintes chamadas de sistema:

| | |
|---|---|
| fork | Cria um processo filho e retorna o valor do seu PID no final da execução |
| wait | Aguarda pela terminação de um processo |
| execlp | Executa um comando no sistema |
| getpid | Retorna o PID do processo corrente |
| getppid | Retorna o PID do pai do processo corrente |
| printf | Imprime texto formatado no terminal |

*Tabela 1 - Programa para criação da árvore de processos* P_A->P_B->P_C.

| Linha | **PRG=arvore-processos; gcc -o $PRG.out $PRG.c; ./$PRG.out** # compile & run |
|---|---|
| 1 | Sou P_A com PID 1028, filho de PID 8 |
| 2 | Eu P_A criei P_B! |
| 3 | Sou P_B com PID 1029, PID interno 0, filho do PID 1028 |
| 4 | Eu P_B criei P_C! |
| 5 | Sou P_C com PID 1030, PID interno 0, filho do PID 1029 |
| 6 | Eu P_C executei: ps |
| 7 | PID TTY TIME CMD<br>8 tty1 00:00:01 bash<br>1028 tty1 00:00:00 q1.out<br>1029 tty1 00:00:00 q1.out<br>1030 tty1 00:00:00 ps |
| 8 | |

| | |
|---|---|
| 9 | Eu P_B aguardei P_C terminar! |
| 10 | Eu P_B executei: ps |
| 11 | PID TTY TIME CMD<br>8 tty1 00:00:01 bash<br>1028 tty1 00:00:00 q1.out<br>1029 tty1 00:00:00 ps |
| 12 | |
| 13 | Eu P_A aguardei P_B terminar! |
| 14 | Eu P_A executei: ps |
| 15 | PID TTY TIME CMD<br>8 tty1 00:00:01 bash<br>1028 tty1 00:00:00 ps |

**(★★)** Crie um programa que simule a árvore de processos da Figura 3.7 embaixo

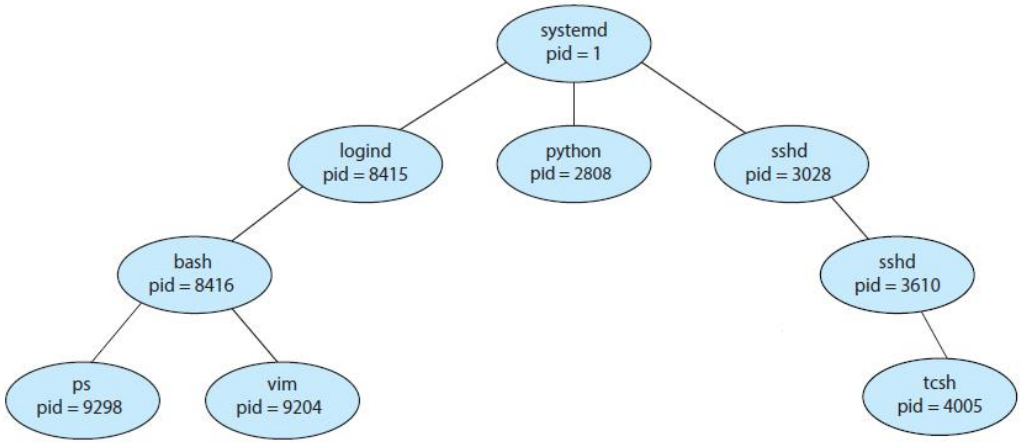


**Figure 3.7** A tree of processes on a typical Linux system.

**(★★)** Escreva o programa “zombie.c” que execute o programa mostrado na Tabela 2. O programa cria um filho e deixa ele como *zombie*, isto é, não aguarda ele terminar. Após um tempo de X segundos o pai executa o comando ps e termina. A saída no terminal apresenta o processo zombie.x no estado <defunct> indicando que zombie.x com PID 1192, filho de PID 1191 virou órfão e, após ser reconhecido como tal pelo SO, virou um “órfão defunto”, isto é, ele foi terminado pelo SO.
Crie um diagrama de estados do processo e explique todos os estados pelos quais passa o processo filho, desde a sua criação até a sua terminação.

- **Sugestões:**
  - Utilize o código fig3-8.c como referência
  - Estude o conteúdo sobre processos no Classroom
  - Revise o manual man das seguintes chamadas de sistema:

sleep(X) Tempo de espera de X segundos

*Tabela 2 - Processo zombie.*

| Linha | **PRG=zombie; gcc -o $PRG.x $PRG.c; ./$PRG.x** # compile & run |
|---|---|
| 1 | PID TTY TIME CMD<br>8 tty1 00:00:02 bash<br>1191 tty1 00:00:00 ps<br>1192 tty1 00:00:00 zombie.x <defunct> |

**(★★)** Escreva o programa “userprog.c” que permita ao usuário ingressar um comando com seus parâmetros e que execute este comando. Faça uso das chamadas fork e execlp.

**(★★★)** Escreva o programa “ordena-array.c” o qual possui uma variável do tipo array contendo dez números desordenados. O processo main deve criar um processo filho usando fork. Em seguida o main deve ordenar o array utilizando o método bubbleSort, enquanto o filho deve utilizar quickSort. Ambos processos deverão mostrar o vetor ordenado por final.

- **Dica:** Utilize a biblioteca de ordenação localizada em “lib/sortlib.c” para chamar ambos métodos de ordenação desde a main.

**(★★)** Crie um diagrama de árvore de processos gerado pela execução de “fork4.c” onde se mostrem todos os processos e seus filhos e discuta o resultado. Como segundo passo, mostre a árvore de processos resultante de utilizar o programa “pstree” e compare ambos os resultados.

**(★)** Descreva a operação de troca de contexto detalhando o ciclo de estados pela qual passa um processo que está sendo preemptado por um outro mais prioritário.