Alexandre Junior Lelis Rodrigues
Matrícula: 20220005736

Código fonte:

```
<stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>

int pidPAI; // Variável global para armazenar o PID do processo pai

// Função para os processos filhos executarem os comandos
void filho(int command) {
  switch (command) {
  case 1:
    printf("\nComando ls:\n");
    execlp("/bin/ls", "ls", NULL);
    break;
  case 2:
    printf("\nComando pwd:\n");
    execlp("/bin/pwd", "pwd", NULL);
    break;
  case 3:
    printf("\nComando whoami:\n");
    execlp("/usr/bin/whoami", "whoami", NULL);
    break;
  case 4:
    printf("\nComando date:\n");
    execlp("/bin/date", "date", NULL);
    break;
  }
}

// Função para os processos pais
int pai(int f) {
  int process;
  if (f == 1) {
    process = 1;
  } else {
    process = 3;
  }

  pid_t pid;
```

```
  pid = fork();
  if (pid < 0) {
    fprintf(stderr, "Fork falhou");
    exit(EXIT_FAILURE); // Em caso de falha, encerra o processo com erro
  } else if (pid == 0) {
    filho(process); // Processo filho executa o comando correspondente
  } else {
    wait(NULL); // Processo pai espera pelo filho
  }

  pid = fork();
  process++;
  if (pid < 0) {
    fprintf(stderr, "Fork falhou");
    exit(EXIT_FAILURE); // Em caso de falha, encerra o processo com erro
  } else if (pid == 0) {
    filho(process); // Processo filho executa o comando correspondente
  } else {
    wait(NULL); // Processo pai espera pelo filho
  }

  return getpid(); // Retorna o PID do processo pai
}

// Função para o processo "avo"
void avo() {
  int process = 1;

  pid_t pid;
  pid = fork();
  if (pid < 0) {
    fprintf(stderr, "Fork falhou");
    exit(EXIT_FAILURE); // Em caso de falha, encerra o processo com erro
  } else if (pid == 0) {
    printf("\nProcesso F%d é filho, PID: %d | PID de seu pai: %d\n", 1,
          pai(process), pidPAI);
    return; // Processo filho encerra após a execução do pai
  } else {
    wait(NULL); // Processo pai espera pelo filho
  }

  pid = fork();
  process++;
  if (pid < 0) {
    fprintf(stderr, "Fork falhou");
    exit(EXIT_FAILURE); // Em caso de falha, encerra o processo com erro
  } else if (pid == 0) {
    printf("\nProcesso F%d é filho, PID: %d | PID de seu pai: %d\n", 2,
```

```
        pai(process), pidPAI);
  } else {
    wait(NULL); // Processo pai espera pelo filho
    printf("\nProcesso P1 é pai, PID: %d\n", getpid());
    printf("\nFim do processo\n");
    sleep(1);
    return; // Processo pai encerra após a execução dos filhos
  }
}

int main(int argc, char *argv[]) {
  pidPAI = getpid(); // Armazena o PID do processo pai
  avo();             // Chama a função para iniciar a execução dos processos
  return EXIT_SUCCESS; // Retorna com sucesso ao finalizar
}
```

Terminal:

```
alexandre@alexandre-1-2:~/code/College/c$ gcc tarefaSO.c -o so
alexandre@alexandre-1-2:~/code/College/c$ ./so

Comando ls:
list  Lists.c  Reverse.c  so  So  stackArray.c  stackLinked.c  tarefaSO.c

Comando pwd:
/home/alexandre/code/College/Ed2/c

Processo F1 é filho, PID: 7747 | PID de seu pai: 7746

Comando whoami:
alexandre

Comando date:
Sat Feb 10 08:26:19 PM -03 2024

Processo F2 é filho, PID: 7750 | PID de seu pai: 7746

Processo P1 é pai, PID: 7746

Fim do processo
```

Recursos utilizados:

Neovim
Lsp: clangd
GCC
Documentações oferecidas pelo professor:
https://docs.google.com/presentation/d/1g1ldB1PJgSOTeTi_V1vZt7UTzbTDtUcv/edit#slide=id.p31