Prática 1 - Sistemas Operacionais

Elton Mauricio Da Silva - RA 11201810955

link de arquivos: <https://github.com/eltonmds/ufabc_operational_systems>

1 Questao 1

Resposta

Na primeira iteração onde o valor de i é igual a 0, é printado o valor de i e o pid do processo original.
Após isso, para cada iteração, é executado o comando fork e é criado um novo processo. Cada processo criado em uma dada iteração executará as iterações restantes, isto é, serão printados o valor de i e do pid do processo.
Como nesse caso, são executadas 2 iterações, tem-se apenas 2 pids printados na tela, do processo original e do processo criado na primeira iteração. Há ainda um terceiro processo criado na segunda iteração, mas como não há iterações restantes para ele executar, não é printado na tela.

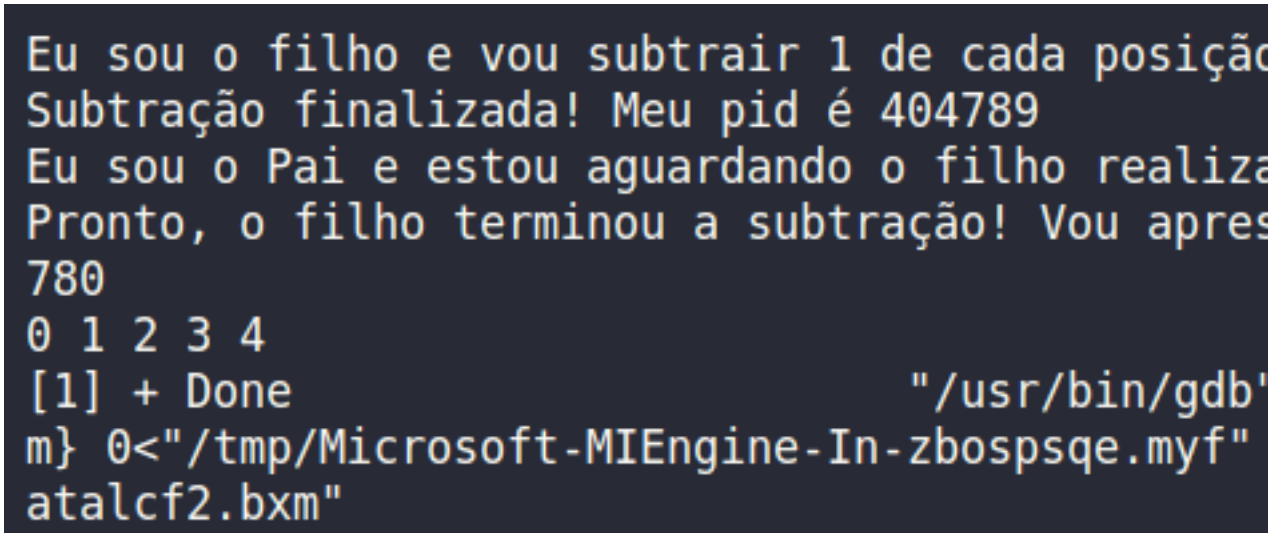
2 Questão 2

Código

```c
#include <sys/wait.h> /* system call - wait */
#include <stdint.h> /* system call - wait */
#include <stdlib.h> /* system call - exit */
#include <unistd.h> /* system call - fork, exec, sleep */
#include <stdio.h>

int G[5] = {0, 1, 2, 3, 4};

void executeSubtraction(int counter) {
    for (counter = 0; counter < 5; counter++) {
        G[counter]--;
    }
}

void printVector(int counter) {
    for (counter = 0; counter < 4; counter++) {
        printf("%d ", G[counter]);
    }
    printf("%d\n", G[counter]);
}

int main(){
    int i;
    pid_t myFork = fork();
    if (myFork > 0) {
        printf("Eu sou o Pai e estou aguardando o filho realizar a subtra  o! Meu pid    %d\n", getpid());
        wait(NULL);
        printf("Pronto, o filho terminou a subtra  o! Vou apresentar o resultado. Meu pid    %d\n", getpid());
        printVector(i);
    }
    else if (!myFork) {
        printf("Eu sou o filho e vou subtrair 1 de cada posi  o do vetor! Meu pid    %d\n", getpid());
        executeSubtraction(i);
        printf("Subtra  o finalizada! Meu pid    %d\n", getpid());
    }
    else if (myFork <= -1) {
        perror("fork");
        exit(EXIT_FAILURE);
```

```
39          }
40
41
42          return 0;
43      }
```

Output



Explicação

O programa primeiro printa a seguinte frase : "Eu sou o Pai estou prestes a ter um filho!". Indicando que nesse momento, quem está rodando é o processo pai e o processo filho ainda não existe. Após isso é utilizado o comando fork, onde é criado um novo processo que é o processo filho.

A partir daí, os dois processos, pai e filho, são concorrentes. O output do programa acontece de acordo com as condições explicitadas no código:

Se o processo que está rodando atualmente é o filho, a seguinte frase é printada: "Eu sou o filho e vou subtrair 1 de cada posição do vetor!", a subtração é executada e é printado: Subtração finalizada!".

Caso o processo que está rodando é o pai, temos esse output: "Eu sou o Pai e estou aguardando o filho realizar a subtração!", e então o processo pai espera pelo término do processo filho.

Depois que o processo filho é finalizado, dentro do processo pai a frase "Pronto, o filho terminou a subtração! Vou apresentar o resultado." é printada. E então, o valor do vetor G é apresentado com os valores originais.

Note que o vetor G foi criado antes do fork, portanto ele tem o mesmo valor para os dois processos nesse momento. No caso do processo filho, é feita a subtração. Então o valor de G muda para -1, 0, 1, 2, 3. Mas não é apresentado, de acordo com o proposto no enunciado.

Já no processo pai, o valor de G apresentado é o original. Pois não houve a subtração, visto que ela só é feita no processo filho.

3 Questão 3

Código

Versão com thread

```c
#include <stdio.h>
#include <unistd.h>
#include <pthread.h>
#include <stdlib.h>
#include <sys/time.h>


int vector[20];
int min, max, mode;
float avg;

void *maxVector() {

    int iterator;
    max = vector[0];
    for (iterator = 1; iterator < 19; iterator++) {
       if (vector[iterator] > max) max = vector[iterator];
    }
    pthread_exit(NULL);
}

void *minVector() {

    int iterator;
    min = vector[0];
    for (iterator = 1; iterator < 19; iterator++) {
        if (vector[iterator] < min) min = vector[iterator];
    }
    pthread_exit(NULL);
}

void *avgVector() {

    int iterator;
    int ans = 0;
    for (iterator = 0; iterator < 19; iterator++) {
        ans += vector[iterator];
    }

```

```
40          avg =  ans/20;
41          pthread_exit(NULL);
42      }
43
44      void *modeVector() {
45
46          int iterator;
47          int maxCount = 0, j;
48          mode = 0;
49          for (iterator = 0; iterator < 20; ++iterator) {
50              int count = 0;
51
52              for (j = 0; j < 20; ++j) {
53                  if (vector[j] == vector[iterator])
54                  ++count;
55              }
56
57              if (count > maxCount) {
58                  maxCount = count;
59                  mode = vector[iterator];
60              }
61          }
62      }
63
64      int main() {
65          struct timeval current_time;
66
67          gettimeofday(&current_time, NULL);
68          printf("seconds : \%ld\nmicro seconds : %ld",
69          current_time.tv_sec, current_time.tv_usec
70          );
71          printf("\n");
72
73
74          pthread_t thr[4];
75
76          for (int i = 0; i < 20; i++) {
77              vector[i] = rand() \% 10 + 1;
78          }
79
80          int i = 0;
81          if(pthread_create(&thr[i], NULL, maxVector, NULL)) {
82              printf("Ops... Houve um erro na cria   o da thread \%d.\n", i)
    ;
83              return 0;
84          }
85          i++;
```

```
86          if(pthread_create(&thr[i], NULL, minVector, NULL)) {
87              printf("Ops... Houve um erro na cria   o da thread \%d.\n", i)
    ;
88              return 0;
89          }
90          i++;
91          if(pthread_create(&thr[i], NULL, avgVector, NULL)) {
92              printf("Ops... Houve um erro na cria   o da thread \%d.\n", i)
    ;
93              return 0;
94          }
95          i++;
96          if(pthread_create(&thr[i], NULL, modeVector, NULL)) {
97              printf("Ops... Houve um erro na cria   o da thread \%d.\n", i)
    ;
98              return 0;
99          }
100
101         (void) pthread_join(thr[0], NULL);
102         (void) pthread_join(thr[1], NULL);
103         (void) pthread_join(thr[2], NULL);
104         (void) pthread_join(thr[3], NULL);
105
106         printf("O valor do vetor   :\n");
107         for (int i = 0; i < 20; i++) {
108             printf("\%d ", vector[i]);
109         }
110         printf("\n");
111
112         printf("O valor m ximo   : \%d.\n", max);
113         printf("O valor m nimo   : \%d.\n", min);
114         printf("O valor da moda   : \%d.\n", mode);
115         printf("O valor da m dia   : \%.2f.\n", avg);
116
117         gettimeofday(&current_time, NULL);
118         printf("seconds : %ld\nmicro seconds : \%ld",
119         current_time.tv_sec, current_time.tv_usec
120         );
121         printf("\n");
122         pthread_exit(NULL);
123     }
```

Versão sem thread

```
1   #include <stdio.h>
2   #include <stdlib.h>
3   #include <sys/time.h>
4
```

```
5     int vector[20];
6     int min, max, mode;
7     float avg;
8
9     void maxVector() {
10        int iterator;
11        max = vector[0];
12        for (iterator = 1; iterator < 19; iterator++) {
13            if (vector[iterator] > max) max = vector[iterator];
14        }
15    }
16
17    void minVector() {
18        int iterator;
19        min = vector[0];
20        for (iterator = 1; iterator < 19; iterator++) {
21            if (vector[iterator] < min) min = vector[iterator];
22        }
23    }
24
25    void avgVector() {
26        int iterator;
27        int ans = 0;
28        for (iterator = 0; iterator < 19; iterator++) {
29            ans += vector[iterator];
30        }
31
32        avg =  ans/20;
33
34    }
35
36    void modeVector() {
37       int iterator;
38       int maxCount = 0, j;
39       mode = 0;
40       for (iterator = 0; iterator < 20; ++iterator) {
41          int count = 0;
42
43          for (j = 0; j < 20; ++j) {
44              if (vector[j] == vector[iterator])
45              ++count;
46          }
47
48          if (count > maxCount) {
49              maxCount = count;
50              mode = vector[iterator];
51          }
```

```
52          }
53
54      }
55
56      int main() {
57          struct timeval current_time;
58
59          gettimeofday(&current_time, NULL);
60          printf("seconds : \%ld\nmicro seconds : \%ld",
61          current_time.tv_sec, current_time.tv_usec
62          );
63          printf("\n");
64
65          for (int i = 0; i < 20; i++) {
66              vector[i] = rand() \% 10 + 1;
67          }
68
69          maxVector();
70          minVector();
71          avgVector();
72          modeVector();
73
74          printf("O valor do vetor   :\n");
75          for (int i = 0; i < 20; i++) {
76              printf("\%d ", vector[i]);
77          }
78          printf("\n");
79
80          printf("O valor m ximo    : \%d.\n", max);
81          printf("O valor m nimo    : \%d.\n", min);
82          printf("O valor da moda   : \%d.\n", mode);
83          printf("O valor da m dia    : \%.2f.\n", avg);
84
85          gettimeofday(&current_time, NULL);
86          printf("seconds : \%ld\nmicro seconds : \%ld",
87          current_time.tv_sec, current_time.tv_usec
88          );
89          printf("\n");
90
91          return 0;
92      }
```

Explicação dos Resultados

Processador da máquina: 11th Gen Intel® Core™ i7-11390H @ 3.40GHz, 4 núcleos, 8 threads O programa sem thread obteve um resultado muito melhor. cerca de 30 vezes

mais rápido. Isso se deve ao fato de que, devido ao tamanho do vetor ser muito pequeno, é mais custoso criar as threads e criar um paralelismo do que rodar mono thread. Para comprovar isso, foi feito um novo teste com um vetor maior, com 20000 itens. Desta vez, o programa com thread obteve um resultado melhor.