Respostas - Threads em C

Link da atividade: ThreadsC/C++.ipynb

2. PosixThreads

1- 210
2- A função soma de 1 até o número informado e substitui o próprio valor pelo resultado da soma.
Com entrada 5 por exemplo temos:
0+1 -> 1+2 -> 3+3 -> 6+4 -> 10+5

3-

```
%%writefile criandothreads.c
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>


int f; // Dado compartilhado entre as threads
bool k;


void *funcaof(void *param); //Assinatura da função que será executada pela
thread.
void *funcaok(void *param);
bool verificaPrimo(int a);


int main(int argc, char *argv[]){
    if (argc != 2 && atoi(argv[1]) < 0) {
      fprintf(stderr,"sintaxe: ./criandothreads <valor inteiro maior que
0>\n");
      return -1;}


    pthread_t thread1; //cria variável do tipo thread
    pthread_attr_t attr; // cria variável do tipo atributo de thread
    pthread_attr_init(&attr); // inicializa attr com valores padroes


    pthread_create(&thread1,&attr,funcaof,argv[1]); // cria a thread


    pthread_t thread2; //cria variável do tipo thread
    pthread_create(&thread2, &attr,funcaok,argv[1]);
```

```
    pthread_join(thread1,NULL);
    pthread_join(thread2,NULL);


    printf("f = %d\n",f);

    printf("eh primo?");
    if(k==1)
      printf("%d é primo",f);
    else
      printf("%d não é primo",f);

}


void *funcaof(void *param) {
    int i, ultimo = atoi(param);
    f = 0;


  if (ultimo > 0)
    for (i = 1; i <= ultimo; i++)
      f += i;
  pthread_exit(0);
}


void *funcaok(void *param){
  int num = atoi(param);
  k = verificaPrimo(num);


}


bool verificaPrimo(int a){
  if(a < 2 ){
    return true;
  }
  else if(a==2){
    return true;
  }
  else if(a%2==0){
    return false;
  }
  else{
    for(int inicio = 3; inicio<= a/2;inicio++){
      if(a/inicio==0)
        return false;
    }
    return true;
```

```
    }
}
```

4- **! unminimize**

5- A função basicamente criará uma nova thread.
6- Os valores associados são:
PTHREAD_SCOPE_PROCESSO: A thread criada não possui vínculo.
PTHREAD_CREATE_JOINABLE: Após o término da thread, ela e a saída são preservados.
NULL: A nova thread tem seu endereço de pilha alocado pelo sistema.
1 megabyte: O sistema define como novo segmento de tamanho da pilha.
Vazio: A nova thread herda a prioridade do thread pai.
PTHREAD_INHERIT_SCHED: A nova thread herda a prioridade de agendamento da thread pai.
SCHED_OTHER: Utilizada a prioridade de agendamento fixa, as thread serão executadas até serem antecipadas por uma thread de prioridade mais alta ou até bloquearem.

7- Essa thread espera que uma outra thread em específico termine primeiro, normalmente a espera é para que seus filhos executem.

3- Threads em modo Kernel

8- Apareceu a tabuada dos números 1 e 2. Com a mudança para 5 6 7 8, foi impresso a tabuada dos 4 números por meio das threads

9- As respostas mudam a cada execução. Isso ocorre no uso de threads, os processos estão executando de forma simultânea e estão misturando as respostas por uma não esperar a outra finalizar.

10- É essa função que executa os cálculos e prints do argumento enviado a ela que por sua vez, é executada por meio de threads, causando a mistura de prints como saída.

11- Clone é uma chamada que cria um novo processo filho, semelhante ao fork.

CLONE_VM: Quando o processo for criado ele deverá compartilhar o mesmo endereço virtual do processo pai.

CLONE_FS: O novo processo deverá compartilhar o mesmo sistema de diretório do processo pai.

CLONE_FILES: O novo processo deverá compartilhar as mesmas entradas de arquivos do processo pai.

CLONE_SIGHAND: Irá compartilhar a mesma tabela de manipuladores de sinais com o processo pai.

12- Cada thread terá o seu stack, em cada uma delas terá a sua sequência de comandos a serem executados que por sua vez, podem ser executados em paralelo com outros, elas não dependem de outros programas.
13- Cada thread terá o tamanho de 64 Kb. É criado o ponteiro que representa a base e topo do stack das threads. Após criado o ID cada thread precisa ter o seu stack alocado que por meio do for finalizado na linha 38 é feito.

14- o processo filho possui valor =0 no pid, ele entra justamente no if que possui a soma de valor+15 e o processo pai tem valor maior que 0, pois agora possui um filho após a execução, muito provavelmente o valor é igual a 1.

15- O valor retornado quando a execução é um sucesso é igual a 0 para o filho e o pid do filho é retornado para o pai.

17- Os números são o ID de cada processo. São 8 números diferentes. 15 processos foram criados com a utilização do fork

18- nps = n^2-1

19-

```
%%writefile 10processos.c
#include <stdio.h>
#include <unistd.h>

int main()
{
        pid_t pid;

        pid = fork();
        fork();
```

```
        if(pid>1)
                fork();
        if(pid>2)
                fork();
        printf("%d\n",getpid());

        return 0;
}
```