Universidade Federal de Roraima
Departamento de Ciência da Computação
Sistemas Operacionais

Atividade - Aula 05/08

Atenção: Vale ressaltar que esta atividade será usada como critério para a contabilização de sua frequência de aula.

Prazo de Entrega: 09/08/2021

Aluno: Pedro Vinícius da Silva Ribeiro MAT:2019033903

1. Conceitue e diferencie threads em modo kernel e usuário.
   Kernel: são mais lentas que no modo usuário, porém tem suporte ao multiprocessamento;

   Usuário: são mais rápidas porque dispensam o acesso ao núcleo;

2. Usando o site www.kernel.org , analise o código fonte da última versão para identificar e apresentar um exemplo do uso threads em modo kernel.

   A criação é feita via kthreadd, para que tenhamos um ambiente limpo mesmo que sejamos invocados do userspace

```
struct kthread {
    unsigned long flags;
    unsigned int cpu;
    int (*threadfn)(void *);
    void *data;
    mm_segment_t oldfs;
    struct completion parked;
    struct completion exited;
#ifdef CONFIG_BLK_CGROUP
    struct cgroup_subsys_state *blkcg_css;
#endif
};
```

3. Utilizando o código disponível (arquivo class_src_0308.zip) no site da disciplina no tópico de aula **Processos: Threads e Modelos Multithreading**, modifique o código em C ou Rust que usa múltiplas thread para utilizar a rotina pthread_join() que espera pelo término de uma thread. Apresente os resultado da execução com e sem o uso da rotina pthread_join()

- **Com JOIN**

```
root@Nitro5:/mnt/d/Download/class_src_0308/class_src/C# ./teste03
No main: criando thread 0
Thread #0!
No main: criando thread 1
Thread #1!
No main: criando thread 2
Thread #2!
No main: criando thread 3
Thread #3!
No main: criando thread 4
Thread #4!
root@Nitro5:/mnt/d/Download/class_src_0308/class_src/C#
```

- **Sem JOIN**

```
root@Nitro5:/mnt/d/Download/class_src_0308/class_src/C# ./teste02
No main: criando thread 0
No main: criando thread 1
Thread #0!
No main: criando thread 2
Thread #1!
No main: criando thread 3
Thread #2!
No main: criando thread 4
Thread #3!
Thread #4!
root@Nitro5:/mnt/d/Download/class_src_0308/class_src/C#
```

4. Escreva um programa na linguagem de programação C utilizando threads para identificar todos os números pares de uma lista com N números.

```c
// 04. Escreva um programa na linguagem de programação C
// utilizando threads para identificar todos os números pares de uma
// lista com N números.

#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <assert.h>
#include <string.h>

// #define N_THREADS 7

enum {EVEN, ODD, ALL};

int type = ALL;

typedef struct block
{
    int n;
    int* numbers;
} Block;

// b   -> Block contendo todos os blocks
// n   -> Quantidade de blocks
void deleteBlock(Block* b, int n){
    int i;
    for(i = 0; i < n; i++)
        free(b[i].numbers);
    free(b);
}

// arr -> array base
// n   -> numero de elementos do array
// m   -> quantidade de subdivisões
Block* subdivideArray(int* arr, int n, int m){
    Block* blocks;
    int i = 0, l = 0, j = 0, _j = 0;
    // quantos elementos são possiveis de ter em
    // cada subarray para que tenhamos m subdivisões.
    int k = (int)(n / m);
    int p = n % m;

    printf("são possiveis %d elementos em cada array, e sobra %d\n", k, p);

    for(i = 0; i < m; i++){

        int* t = (int*)malloc(k*sizeof(int));
        if(i == 0){ // cria o primeiro block
            blocks = (Block*)malloc(sizeof(Block));
        }else{      // realoco i+1 na heap
            blocks = (Block*)realloc(blocks, (i+1)*sizeof(Block));
        }

        blocks[i].n = k;
        l = 0;
        for(j = _j; j < _j+k; j++)
            t[l++] = arr[j];
        _j = j;
        blocks[i].numbers = t;

```

```
61      }
62
63      // Os valores que sobrarem resultante da subdivisão
64      // que sairam da margem do calculo, serão colocados
65      // nos blocks existentes a partir do primeiro
66      while(_j < n){
67          for(i = 0; i < m; i++){
68              if(_j > n-1) break;
69              Block* _t = &blocks[i];
70              _t->n++;
71              _t->numbers = (int*)realloc(_t->numbers, _t->n*sizeof(int));
72              _t->numbers[_t->n-1] = arr[_j++];
73          }
74      }
75
76      return (blocks);
77
78  }
79
80  void description(int n){
81      if((n % 2) == 0 && (type == ALL || type == EVEN))
82          printf("%d[P]\t", n);
83      if((n % 2) != 0 && (type == ALL || type == ODD))
84          printf("%d[I]\t", n);
85  }
86
87  void* isEven(void* blocks){
88      int i;
89      Block* b = (Block*)blocks;
90      for(i = 0; i < b->n; i++){
91          int n = b->numbers[i];
92          description(n);
93      }
94      printf("\n");
95      pthread_exit(NULL);
96  }
97
98  int main(int argc, char *argv[]){
99      int i, j, k, error, N_THREADS, SIZE;
100     int* arr = NULL;
101     // thread configs
102     int join = 0;
103     k = 0;
104     for(i = 0; i < argc; i++){
105         k++;
106         if(strcmp(argv[i], "-r")==0)
107         {
108             SIZE = (rand()%90)+10;
109             arr = (int*)malloc(SIZE*sizeof(int));
110             printf("Aleatoriedade de N[%d] numeros!\n", SIZE);
111             for(j = 0; j < SIZE; j++)
112                 arr[j] = rand()%120;
113         } else if(strcmp(argv[i], "-n")==0) {
114             SIZE = (argc-k);
115             arr = (int*)malloc(SIZE*sizeof(int));
116             for(i = k; i < argc; i++)
117                 arr[i-k] = atoi(argv[i]);
118         } else if(strcmp(argv[i], "-j")==0){
119             join = 1;
120         } else if(strcmp(argv[i], "-even")==0){
```

```c
            type = EVEN;
        } else if(strcmp(argv[i], "-odd")==0){
            type = ODD;
        } else if(strcmp(argv[i], "-all")==0){
            type = ALL;
        }

    }


    if(!(N_THREADS=atoi(argv[1]))){
        printf("N_THREADS não pode ser 0, ou não foi possivel converter!\n");
        exit(-1);
    }

    // numero de threads deve ser menor que o numero de elementos
    assert(N_THREADS ⩽ SIZE);

    printf("N_THREADS: %d\n", N_THREADS);

    pthread_t threads[N_THREADS];

    Block* b = subdivideArray(arr, SIZE, N_THREADS);
    for(i = 0; i < N_THREADS; i++){
        error = pthread_create(&threads[i], NULL, isEven, (void*)(b+i));
        assert(error == 0);
        if(join)pthread_join(threads[i], NULL);
    }
    pthread_exit(NULL);
    return 0;

}
```