Atividade da Aula 05 (08/09/2021)

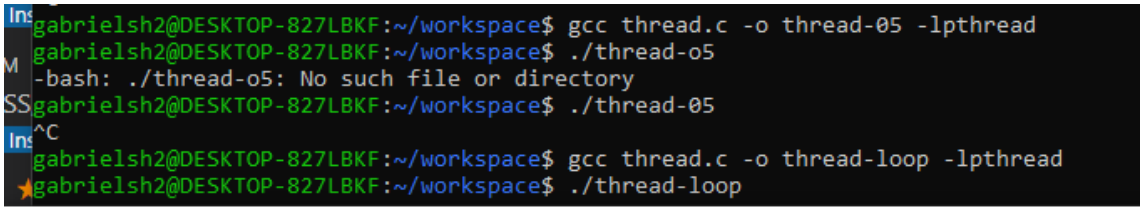
Aluno: Gabriel Hoffmann

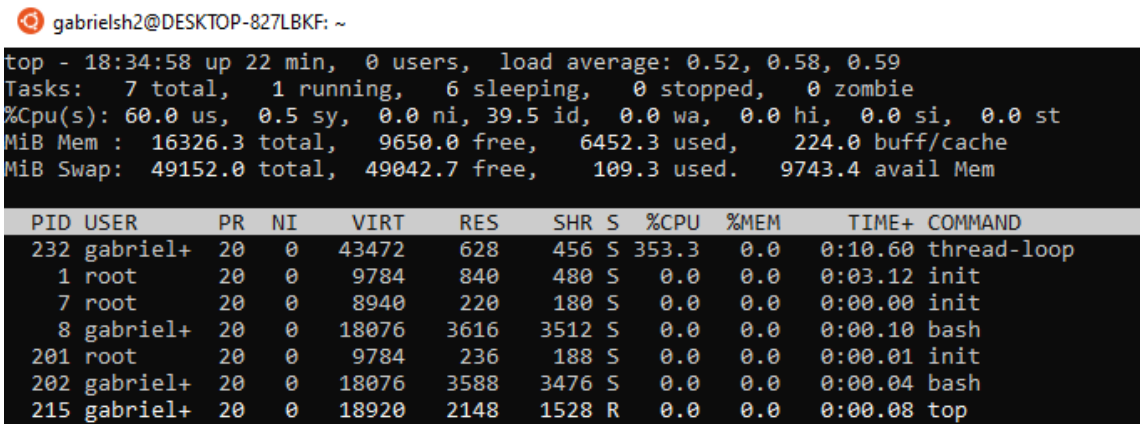
Curso: Ciência da Computação

Refaça o exercício apresentado em aula, modificando para tornar as threads em loop e descreva os comportamentos considerando os comandos:

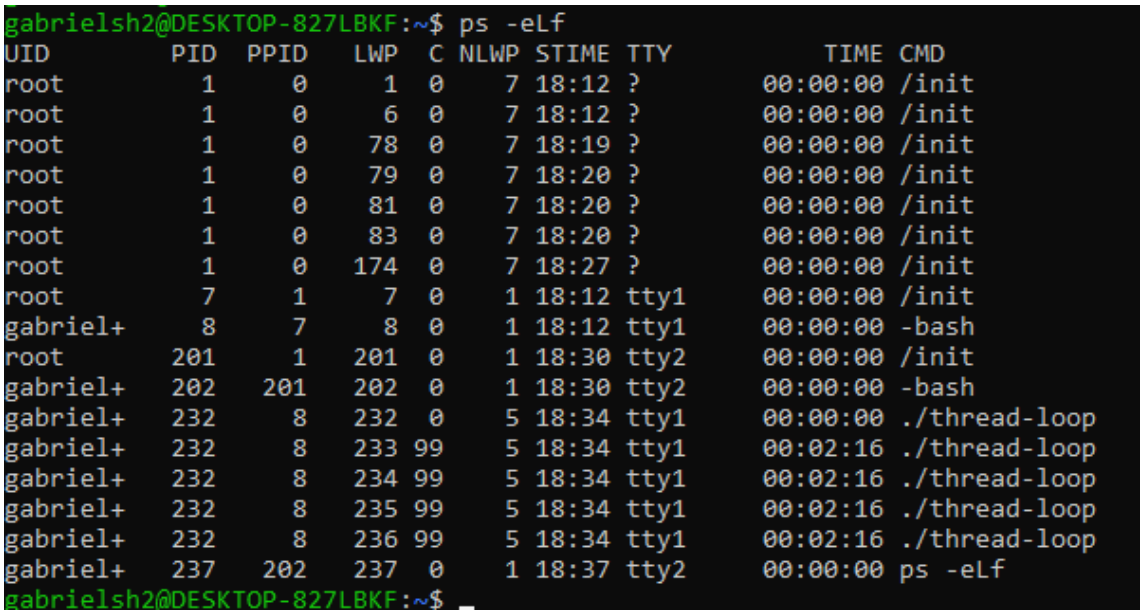
```c
#include <pthread.h>
#include <stdio.h>
unsigned long soma[4];
void *thread_fn(void *arg)
{
    long id = (long)arg;
    int inicio = id * 2500000000;
    int i = 0;
    while (i < 2500000000)
    {
        soma[id] += (i + inicio);
        i++;
    }
    return NULL;
}
int main(void)
{
    pthread_t t1, t2, t3, t4;
    unsigned long resultado = 0;
    pthread_create(&t1, NULL, thread_fn, (void *)0);
    pthread_create(&t2, NULL, thread_fn, (void *)1);
    pthread_create(&t3, NULL, thread_fn, (void *)2);
    pthread_create(&t4, NULL, thread_fn, (void *)3);
    pthread_join(t1, NULL);
    pthread_join(t2, NULL);
    pthread_join(t3, NULL);
    pthread_join(t4, NULL);
    resultado = soma[0] + soma[1] + soma[2] + soma[3];
    printf("%lu\n", resultado);
    return 0;
}
```

Aumentei o laço de repetição para o programa entrar em loop e poder a partir de outro terminal verificar seu funcionamento com outros comandos





Utilizando o top podemos observar o programa rodando em loop com o PID 232, além de outros dados disponibilizados pelo top como o tempo de execução.



Com o comando ps –eLf podemos ver as threads separadas rodando no mesmo tempo de 2 minutos e 16 segundos, além de outros dados sobre o processo.

Entrando no /proc do processo na pasta do seu PID temos acesso a vários outros dados

```
gabrielsh2@DESKTOP-827LBKF:/proc/232$ ls
attr    cmdline  environ  gid_map  mountinfo  net      oom_score_adj  setgroups  statm   uid_map
auxv    comm     exe      limits   mounts     ns       root           smaps      status
cgroup  cwd      fd       maps     mountstats oom_adj  schedstat      stat       task
```

Um exemplo dos diversos dados que podem ser lidos, usando o vim no status temos informações detalhadas do processo onde podemos ver seu número de threads ou o limits em que temos informações sobre as limitações do quanto a máquina disponibilizou para o processo.

```
Name:   thread-loop
State:  S (sleeping)
Tgid:   232
Pid:    232
PPid:   8
TracerPid:      0
Uid:    1000    1000    1000    1000
Gid:    1000    1000    1000    1000
FDSize: 3
Groups:
VmPeak: 0 kB
VmSize: 43472 kB
VmLck:  0 kB
VmHWM:  0 kB
VmRSS:  628 kB
VmData: 0 kB
VmStk:  0 kB
VmExe:  4 kB
VmLib:  0 kB
VmPTE:  0 kB
Threads:        5
SigQ:   0/0
SigPnd: 0000000000000000
ShdPnd: 0000000000000000
SigBlk: 0000000000000000
SigIgn: 0000000000000000
SigCgt: 0000000000000000
"status" [readonly] 37L, 660C
```

```
Limit                     Soft Limit           Hard Limit           Units
Max cpu time              unlimited            unlimited            seconds
Max file size             unlimited            unlimited            bytes
Max data size             unlimited            unlimited            bytes
Max stack size            8388608              unlimited            bytes
Max core file size        0                    unlimited            bytes
Max resident set          unlimited            unlimited            bytes
Max processes             8041                 8041                 processes
Max open files            1024                 4096                 files
Max locked memory         65536                65536                bytes
Max address space         unlimited            unlimited            bytes
Max file locks            unlimited            unlimited            locks
Max pending signals       8041                 8041                 signals
Max msgqueue size         819200               819200               bytes
Max nice priority         40                   40
Max realtime priority     0                    0
Max realtime timeout      unlimited            unlimited            us
```

Exercícios teóricos

A correta utilização de processos e *threads* é fundamental para garantir o desempenho e a transparência de sistemas distribuídos. Sobre esse tema, considere as afirmativas a seguir.

I. A sobreposição de *threads* em um processo é o principal recurso para obtenção de alto grau de transparência de distribuição em redes com longos tempos de propagação de mensagens.

II. A desvantagem de se estruturar um programa para utilizar múltiplas *threads* é que ele ficará dependente de sistemas multiprocessadores.

III. O modelo de *threads* implementado pelo sistema operacional deve ser aquele em que o gerenciamento de *threads* fica inteiramente no espaço de cada processo para evitar trocas de contexto entre processos e o núcleo (*kernel*) no chaveamento de *threads*.

IV. Servidores *multithreaded* têm melhor desempenho se estruturados com ao menos uma *thread* despachante e várias *threads* operárias para recebimento e processamento de requisições.

Assinale a alternativa correta.

b) Apenas I e IV estão corretas

Considere o seguinte programa com dois processos concorrentes. O escalonador poderá alternar entre um e outro, isto é, eles poderão ser intercalados durante sua execução. As variáveis x e y são compartilhadas pelos dois processos e inicializadas antes de sua execução.

```
programa P
int x = 0;
int y = 0;
processo A {
      while (x == 0);
      print(‘‘a’’);
      y = 1;
      y = 0;
      print(‘‘d’’);
      y = 1;
}
```

```
processo B {
     print(‘‘b’’);
     x = 1;
     while (y == 0);
     print("c");
}
```

As possíveis saídas são:

(b) badc ou bacd