Universidade Federal de Uberlândia
Disciplina: Sistemas Operacionais (SO)
Prática: Threads

Threads em Ambiente LINUX

Essa atividade está dividida em duas partes, sendo que na Parte 1 os alunos devem explorar o threads em ambiente do Linux. São apresentados comandos para que os estudantes possam se familiarizar com as principais funções empregadas para manipulação dos threas em POSIX. Na Parte 2, será aplicado um questionário com questões objetivas sobre os tópicos abordados em nossa disciplina.

Parte 1: Threads

Threads

Como vimos em aula teórica de thread, um único processo pode ter vários fluxos de execução, que compartilham uma mesma área de código e dados. Cada thread tem de forma independente seu próprio contexto, pilha e contador de programa (program counter – PC).

Nesse atividade será empregado a biblioteca conhecida como *POSIX Threads* ou p*threads*. Um exemplo simples de um processo com múltiplos threads é apresentado na Figura 1.

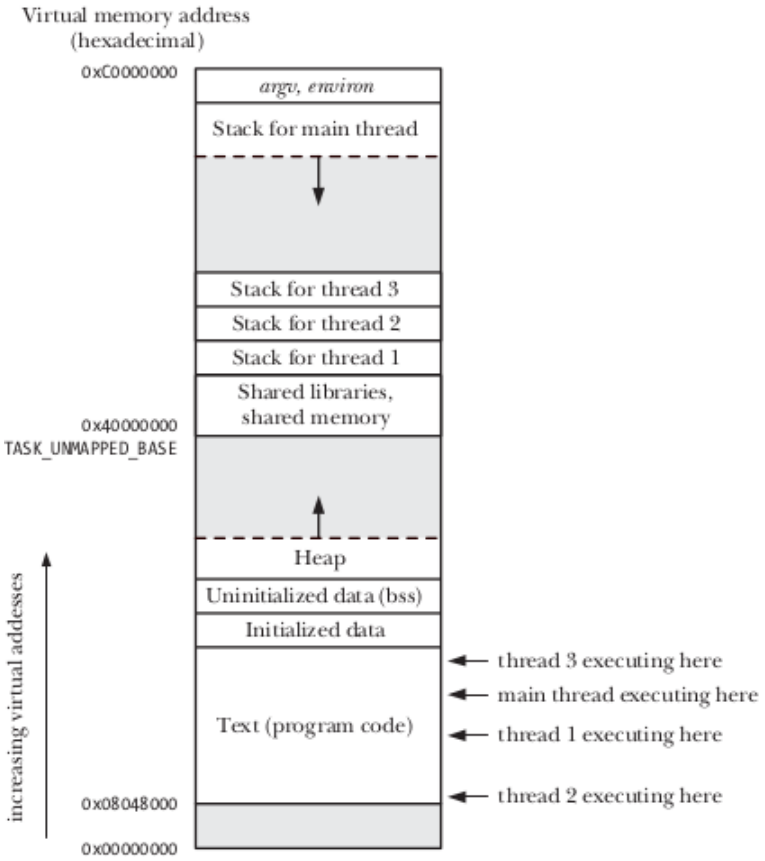

Figura 1 – Um simples exemplo de um processo com 4 threads em execução (Linux / x86-32)

Todo programa em linguagem C que usa *pthreads* deve incluir no cabeçalho a biblioteca pthreads "**#include <pthread.h>"** no início do arquivo.

Para compilar os programas construídos em linguagem C, você deve digitar "**gcc -o nome_do_aplicativo nome_do_programa.c".** Se estiver trabalhando com a biblioteca pthread deve incluir "**-pthread"** na compilação (**$ gcc -o threadExemplo threadExemplo.c -pthread**).

Criando e terminando uma thread

O padrão pthread exige que funções que serão chamadas para a criação de novas threads possuam uma assinatura específica "**void* (void*)**". A função a ser executada precisa obrigatoriamente retornar um ponteiro genérico e receber como parâmetro de entrada um ponteiro genérico. O parâmetro de entrada pode ser usado para passar um dado qualquer para o novo thread.

Para criar o *thread* usa-se a função *pthread_create(*t, *a, rotina, arg)*, onde os argumentos são:

- *t* é um ponteiro para uma variável do tipo *pthread_t* que conterá o identificador da *thread* recém criada;
- *a* é um ponteiro para os atributos do *thread*. Os atributos são armazenados em uma variável do tipo *pthread_attr_t*. Um valor *NULL* indica o uso de valores default. Para detalhes veja pthread_attr_init.
- *rotina* é o nome (ponteiro) para a rotina (função) que define o comportamento do *thread*.
- *arg* é um *void* * que é passado como argumento para a rotina. Recomendo executar "man pthread_create" e dar uma breve lida.

Exemplo 1

```
a)  #include <stdio.h>
b)  #include <pthread.h>
c)
d)  void *OLA(void *argumentos)    {
e)     printf("\nOLÁ MUNDO.. BEM VINDO :-)\n\n");
f)  }

g)  int main ( ){
h)    pthread_t thread;
i)    int flag, i;
j)    printf("criar uma thread\n");
k)    flag = pthread_create(&thread, NULL, OLA, NULL);

l)    if (flag!=0)
m)       printf("Erro na criação da thread\n");
n)    pthread_exit(NULL);
o)    return 0;
p)  }
```

A chamada à função **pthread_exit()** provoca a finalização do thread e a libertação dos recursos utilizados.

1) Escreva e compile o programa do **Exemplo 1**.
   a) Execute e descreva quantos threads foram criados na execução desse código?
   b) Quantas mensagens aparecem na tela? Por que isso acontece?

Esperando por um thread

A função **pthread_join()** permite que um thread espere pela finalização de uma thread específica.

Exemplo 2

```
1.  #include <stdio.h>
2.  #include <pthread.h>
3.  #define NUM_THREADS     10
4.
5.  void *imprime(void *argumentos){
6.     printf("\nUFU...BSI018\n\n");
7.  }
8.  int main (){
9.    pthread_t threads[NUM_THREADS];
10.   int i;
11.   for(i=0;i < NUM_THREADS;i++)
12.        pthread_create(&threads[i], NULL, imprime, NULL);
13.
14.   printf("Espera a finalização das threads criadas \n");
15.
16.   for(i=0; i < NUM_THREADS;i++)
17.      pthread_join(threads[i],NULL);
18.
19.   return 0;
20. }
```

2) Compile o programa **Exemplo 2** e depois execute-o. Quantos threads foram criadas. Qual a finalidade do segundo parâmetro da chamada **pthread_join**?

Passagem de argumentos para o thread

A rotina pthread_create() permite passar argumentos a função de um thread: **pthread_create(&threads[i], NULL, funcao, &i)**.

```
1.  void *imprime ( void * argumentos ){
2.      int valor =* (int *) argumentos;
3.     printf("Valor: %d \t", valor );
4.  }
```

3) Crie um novo programa baseado no **Exemplo 2** e utilize agora a passagem de parâmetro para a função Imprime do código. Utilize o valor da variável da estrutura de repetição, i, para a passagem de parâmetro a função imprime. Compile o programa e depois execute-o. Mostre o resultado dessa operação. Os valores da variável i foram apresentados de forma sequencial?

Thread versus fork

O programa abaixo mostra o uso da biblioteca Pthread em sistema Linux.

Exemplo 3

```
1.  #include <pthread.h>
2.  #include <stdio.h>
3.  #include <stdlib.h>
4.
5.  #define NTHREADS 500
6.
7.  void *do_nothing(void *null) {
8.  int i;
```

```
9.  i=0;
10. pthread_exit(NULL);
11. }
12.
13. int main(int argc, char *argv[]) {
14. int rc, i, j, detachstate;
15. pthread_t tid;
16. pthread_attr_t attr;
17.
18. pthread_attr_init(&attr);
19. pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_JOINABLE);
20.
21. for (j=0; j<NTHREADS; j++) {
22.   rc = pthread_create(&tid, &attr, do_nothing, NULL);
23.   if (rc) {
24.     printf("ERROR; return code from pthread_create() is %d\n", rc);
25.     exit(-1);
26.     }
27.
28.   /* Wait for the thread */
29.   rc = pthread_join(tid, NULL);
30.   if (rc) {
31.     printf("ERROR; return code from pthread_join() is %d\n", rc);
32.     exit(-1);
33.     }
34.   }
35.
36. pthread_attr_destroy(&attr);
37. pthread_exit(NULL);
38.
39. }
```

Programa para criação de threads com a Pthread.

4) Use o código disponível no moodle (exemplo3.c). Compile e execute o código para responder as questões:

a) Execute o programa com valor da variável NTHREADS = 500. Avalie o tempo total de processamento. Repita o experimento 10 vezes e calcule o tempo médio de processamento. Para obter o tempo de processamento no Linux, utilize o comando **time** que irá mostrar o tempo em modo usuário e modo kernel.

Exemplo: **$time ./exemplo3**

b) Repita o item a) para **NTHREADS** = 5.000 e 50.000.

c) Utilize o código do **Exemplo 3** e modifique esse programa para trabalhar com a função **fork.** Faça uma comparação de desempenho entre o **fork** e **pthread** em relação ao tempo de processamento para os diversos valores (500, 5.000 e 50.000). Você conseguiu executar em todas as situações o aumento do número de threads e processos? Como o parâmetro tempo foi afetado com o aumento de invocação das funções (pthread.create e fork)?

5) No **exercício 2.1(d)** fizemos uso do código **fork4.c** Os processos pai e filho são executados separadamente e cada um chama a função **adjustX()** com parâmetros diferentes em cada processo. Agora esse código foi adaptado para usar threads. Vamos examinar o código-fonte e tentar determinar o que ele faz. A saída é semelhante ao fork4.c? Enquanto o programa roda, execute (em outro terminal) o comando **ps xl**. Quantos processos temos nesse caso? Investigue o efeito de remover o *loop* infinito no fim do **main()**. O que acontece? Por que?