Universidade Federal do ABC
Disciplina: BC 1518 – Sistemas Operacionais (SO)
Prática: Threads – Atividade 3

Threads em Ambiente LINUX

Objetivo

Estudo e testes com a biblioteca *pthreads*

Threads

Como vimos em aula teórica, um único processo podemos ter vários fluxos escalonáveis de execução, que compartilham uma mesma área de código e dados, mas cada qual tem de forma independente seu próprio contexto, pilha e program counter (PC). Nesse item, faremos uso da biblioteca conhecida como *POSIX Threads* ou popularmente *pthreads*. Todo programa em C que usa *pthreads* deve incluir o cabelhaço 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 "**-lpthread"** na compilação.

1.Criando e terminando um thread

O padrão PThreads 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 a nova thread.

Para criar uma *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 da *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 da *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

```
1. #include <stdio.h>
2. #include <pthread.h>
3.
4. void *OLA(void *argumentos)    {
```

```
5.      printf("\nOLÁ UFABC... BEM VINDO :-)\n\n");
6. }
7.
8. int main ( ) {
9.   pthread_t thread;
10.     int flag, i;
11.
12.     printf("criar uma nova thread\n");
13.     flag = pthread_create(&thread, NULL, OLA, NULL);
14.
15.     if (flag!=0)
16.        printf("Erro na criação da thread\n");
17.     pthread_exit(NULL);
18.     return 0;
19.   }
```

A chamada à função pthread_exit() provoca a finalização da thread e a libertação dos recursos que estava utilizando. Escreva programa do exemplo 1. Compile o programa (gcc -o thread thread.c -lpthread) e depois execute-o. Quantos threads são criados. Quantas mensagens aparecem na tela?

2. Esperando as threads

A função rotina pthread_join() permite que uma thread espere pela finalização de 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("\nUFABC...BC1518\n\n");
7. }
8.
9. int main (){
10.       pthread_t threads[NUM_THREADS];
11.       int i;
12.
13.       for(i=0;i < NUM_THREADS;i++)
14.            pthread_create(&threads[i], NULL, imprime, NULL);
15.
16.       printf("Espera a finalização das threads criadas \n");
17.
18.       for(i=0;i < NUM_THREADS;i++)
19.           pthread_join(threads[i],NULL);
20.
21.       return 0;
22.     }
```

Compile o programa e depois execute-o. Quantos threads são criados. Quantas mensagens aparecem na tela?

3. Passagem de Argumentos para threads

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. }
```

Crie um novo programa e utilize a passagem de parâmetro. Utilize o valor da variável da estrutura de repetição para passagem de parâmetro. Compile o programa e depois execute-o. Quantos threads são criados?

4. O uso de Fork versus pthreads

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

```
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.  }
```

Construa um programa em linguagem C para execução do código acima. Responda 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. Exemplo: **$ time ./fork**

b) Repita o item a) para NTHREADS = 5000 e 50000.

c) Utilize a atividade 2 (exercício 3) e faça uma comparação de desempenho entre o fork e threads.

4. Exercícios

4.1 - Construa um programa em C que calcule e imprima a soma dos N primeiros números naturais. O usuário deve entrar com o valor numérico. Construa uma função que calcule a soma e utilize threads para realizar essa operação.

4.2 - Para avaliar o comportamento multithreading, implemente o programa abaixo. Uma vez executado faça a experiência de remover a linha de código *error = pthread_join(tid,NULL);* e avalie novamente comportamento do programa. Em seguida, repita o procedimento mas substitua o código supracitado por *error = pthread_detach(tid,NULL).* O que ocorre se modificarmos o valor da função sleep para 10. Utilieza a chamada time para capturar o tempo de execução. Por exemplo: $ time ./thread.

```
#include <stdio.h>
#include <pthread.h>
int count = 0;
void work(void) {
   pthread_t tid;
   tid = pthread_self();
   printf("I am a worker thread %d - count = %d\n", (int) tid, count++);
  sleep(1);
}
int main() {
   int error,i,n;
   pthread_t tid,mytid;
   printf("Enter number of threads: ");
   scanf("%d",&n);
   mytid = pthread_self();
```

```
   printf("thread %d is creating threads\n",(int)     mytid);

   for (i=1;i<=n;i++) {
    error = pthread_create(&tid, NULL,(void *(*)(void *))work, NULL);
    printf("created thread %d\n",(int) tid);
    error = pthread_join(tid,NULL);
    }
    printf("Done forking and joining threads\n");
    return 0;
}
```