Threads

PThreads

- Posix threads
  - Compatibilidade: suporte na maioria dos sistemas operacionais
- Operações básicas
  - Criação
  - Finalização
  - Sincronização (joins,blocking)
  - Escalonamento
  - Gerenciamento de dados internos

Criar threads

- int pthread_create(pthread_t *thread, const pthread_attr_t *attr, void *(*start_routine)(void*), void *arg);
  - Thread: identificador da thread
  - Attr: atributos para modificar o comportamento da thread (NULL, inicializa com o valor padrão)
  - Start_routine: ponteiro para a função de início da thread
  - Arg: passagem de parâmetros.

Join

- Pthread_join(pthread_t thread, void **value_ptr)
  - Espera até que a thread seja finalizada
  - Value_ptr, o valor de retorno da função executando a thread

Exemplo

```
1.    #include <pthread.h>
2.    #include <stdio.h>
3.    #include <linux/unistd.h>
4.    #include <sys/types.h>
5.    #include <syscall.h>

6.    void *thread_function(){
7.      int i;
8.      int tid;
9.      tid= syscall(SYS_gettid);
10.     for (i=0; i < 20; i++){
11.      printf(" thread pid %d %d\n", tid, i);
12.      usleep(10);
13.     }
14.     return NULL;
15.   }

16.   int main()
17.   {
18.      pthread_t p1, p2;

19.      pthread_create(&p1, NULL, &thread_function, NULL);
20.      pthread_create(&p2, NULL, &thread_function, NULL);

21.      pthread_join(p1, NULL);
22.      pthread_join(p2, NULL);
23.      return 0;
24.   }
```

Exemplo(2)

```
1.    #include <pthread.h>
2.    #include <stdio.h>
3.    #include <linux/unistd.h>
4.    #include <sys/types.h>
5.    #include <syscall.h>
6.    int i;

7.    void *thread_function(){
8.      int tid;
9.      tid= syscall(SYS_gettid);
10.     for (; i < 20; i++){
11.      printf(" thread pid %d %d\n", tid, i);
12.      usleep(10);
13.     }
14.     return NULL;
15.   }

16.   int main()
17.   {
18.      pthread_t p1, p2;
19.      i= 0;
20.      pthread_create(&p1, NULL, &thread_function, NULL);
21.      pthread_create(&p2, NULL, &thread_function, NULL);

22.      pthread_join(p1, NULL);
23.      pthread_join(p2, NULL);
24.      return 0;
25.   }
```

Informática

Sincronização

- Duas formas:
  - Mutex
  - Semáforos

Mutex

```
1.    #include <pthread.h>
2.    #include <stdio.h>
3.    #include <linux/unistd.h>
4.    #include <sys/types.h>
5.    #include <syscall.h>

6.    pthread_mutex_t mutex=PTHREAD_MUTEX_INITIALIZER;
7.    void *thread_function(){
8.      int tid;
9.      int i;
10.     tid= syscall(SYS_gettid);
11.     pthread_mutex_lock(&mutex);
12.     for (i=0; i < 20; i++){
13.      printf(" thread pid %d %d\n", tid, i);
14.      usleep(10);
15.     }
16.     pthread_mutex_unlock(&mutex);
17.     return NULL;
18.   }

19.   int main()
20.   {
21.      pthread_t p1, p2;
22.      pthread_create(&p1, NULL, &thread_function, NULL);
23.      pthread_create(&p2, NULL, &thread_function, NULL);

24.      pthread_join(p1, NULL);
25.      pthread_join(p2, NULL);
26.      return 0;
27.   }
```

Informática

Semaféros

- sem_init(sem_t sem, int pshared, unsigned int value);
  - Pshared=0 somente threads, != 0 interprocessos
- sem_wait(sem_t);
- sem_post(sem_t);

Exemplo

```
1.    #include <pthread.h>
2.    #include <stdio.h>
3.    #include <linux/unistd.h>
4.    #include <sys/types.h>
5.    #include <syscall.h>
6.    #include <semaphore.h>

7.    sem_t semid;
8.    void *thread_function(){
9.      int tid;  int i;
10.     tid= syscall(SYS_gettid);
11.     sem_wait(&semid);
12.     for (i=0; i < 20; i++){
13.      printf(" thread pid %d %d\n", tid, i);
14.      usleep(10);
15.     }
16.     sem_post(&semid);
17.     return NULL;
18.   }

19.   int main(){
20.      pthread_t p1, p2;

21.      sem_init(&semid, 0, 1);
22.      pthread_create(&p1, NULL, &thread_function, NULL);
23.      pthread_create(&p2, NULL, &thread_function, NULL);

24.      pthread_join(p1, NULL);
25.      pthread_join(p2, NULL);
26.      sem_destroy(&semid);
27.      return 0;
28.   }
```

Exercícios

- a) Faça um programa seqüencial e outro multithread para realizar a busca de um elemento em um vetor de inteiros
- b) Implemente o problema do produtor-consumidor com um buffer limita utilizando threads