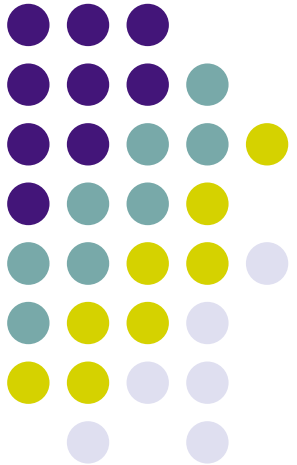
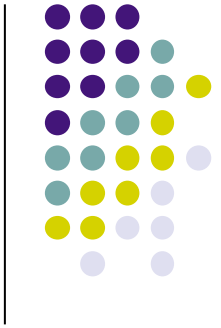
Sistemas de Computação

Threads



Criação de Threads



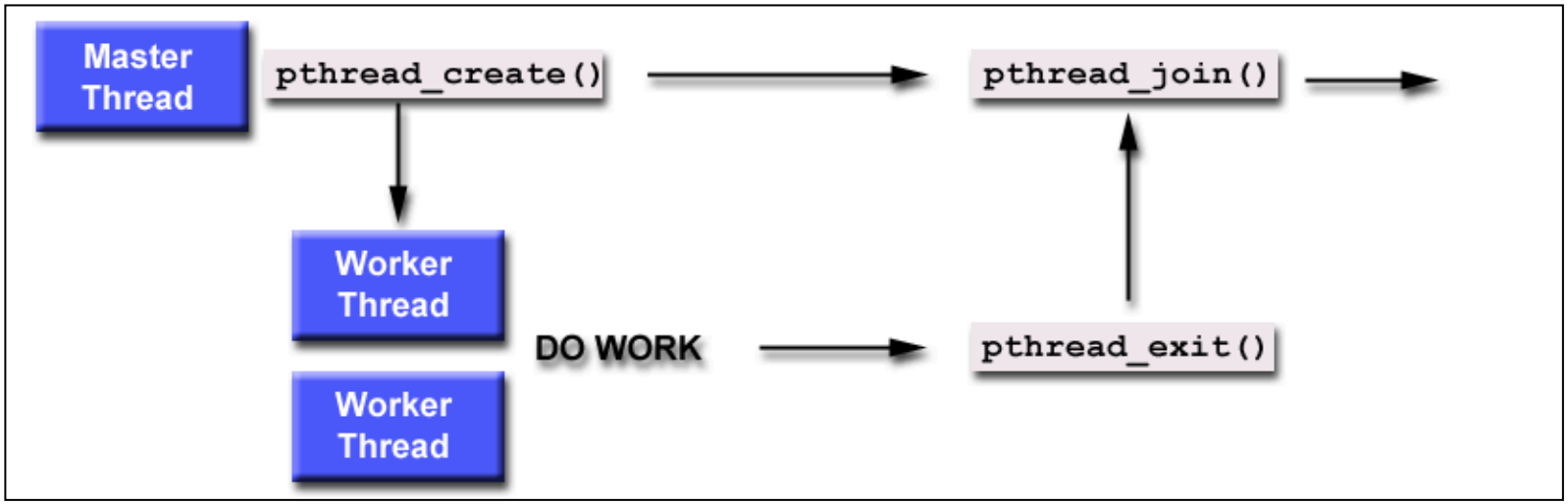
```
int pthread_create(pthread_t * thread,
                   const pthread_attr_t * attr,
                   void * (*start_routine)(void *),
                   void *arg);
```

- `thread` - retorna a threadID
- `attr` – define vários parâmetros de escalonamento. Use NULL se os atributos default devem ser usados. Senão modifique elementos da struct `pthread_attr_t`, definida em pthreadtypes.h
- start_routine é ponteiro para procedimento a ser executado pela thread
- `arg` – ponteiro para argumento do procedimento. Caso sejam muitos argumentos, passe um ponteiro para um struct
- Retorna valor diferente de 0 se criação foi bem sucedida

Sincronismo entre Threads

int pthread_join(pthread_t tid, void* status)
// a thread invocadora é bloqueada até que a thread tid termine

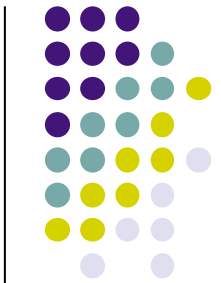
- tid A threadID pela qual deseja-se esperar;
- status O valor de retorno da thread execurando o exit(), será copiado para status




```
void main() {
        pthread_t tid;
        int status;
              pthread_create(&tid,NULL,thread_main,NULL);
        ….
        pthread_join(tid, (void*) &status);
        printf("Return value is: %d\n", status);
}
```

```
void *thread_main( ){
        int result;
        ....
        Pthread_exit((void*) result);
}
```

Exemplo de Uso de Threads



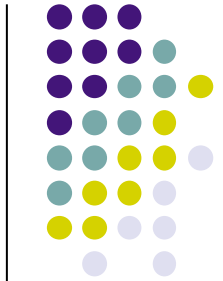
```
#include <pthread.h>
#include <stdio.h>
#define NUM_THREADS 5

void *PrintHello(void *threadid)
{
        printf("\n%d: Hello World!\n", threadid);
        /* do other things */
        pthread_exit(NULL);             /*not necessary*/
}

int main()
{
        pthread_t threads[NUM_THREADS];
        int t;
        for(t=0;t < NUM_THREADS;t++)
        {
          printf("Creating thread %d\n", t);
          pthread_create(&threads[t], NULL, PrintHello, (void *)t);
        }

        for(t=0; t < NUM_THREADS; t++)
          pthread_join(threads[t],NULL);        /* wait for all the threads to
                                                   terminate*/
}
```

Garantindo Exclusão mútua



pthread_mutex_t m = PTHREAD_MUTEX_INITIALIZER;

int pthread_mutex_init (pthread_mutex_t *mutex,
const pthread_mutexattr_t *mutexattr);

int pthread_mutex_lock (pthread_mutex_t *mutex);
int pthread_mutex_unlock (pthread_mutex_t *mutex);
int pthread_mutex_destroy (pthread_mutex_t *mutex);

```
void *thread_function(void *);
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
int  counter = 0;

void *thread_function(void *dummyPtr)
{
   printf("Thread number %ld\n", pthread_self());
   pthread_mutex_lock( &mutex );
   counter++;
   pthread_mutex_unlock( &mutex );
}
```

Variável de condição

É uma variável do tipo **pthread_cond_t** usada para suspender a execução da thread até que certa condição esteja satisfeita.

Precisa estar sempre associada a um mutex.

Exemplo:

```
pthread_cond_t can_produce;

can_produce = PTHREAD_COND_INITIALIZER;

if(buffer->len == BUF_SIZE) { // full
        // wait until some elements are consumed
        pthread_cond_wait(&can_produce, &mutex);

pthread_cond_signal (&can_produce);
```

Corrida dos Sapos

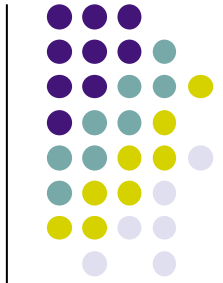

```
#define NUM_THREADS      5
#define PULO_MAXIMO      100
#define DESCANSO_MAXIMO  1
#define DISTANCIA_PARA_CORRER
static int classificacao = 1;
static pthread_mutex_t lock;
static char * resp[200];

static int cont = 0;int main()
{
 classificacao =1;
 pthread_t threads[NUM_THREADS];
 int t;
 printf("Corrida iniciada ... \n");

 for(t=0;t < NUM_THREADS;t++)  pthread_create(&threads[t], NULL, Correr, (void *) t);
 for(t=0;t < NUM_THREADS; t++)  pthread_join(threads[t],NULL)

 printf("\n Acabou!!\n");
 pthread_exit(NULL);
}
```

Corrida dos Sapos

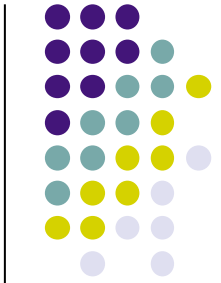


```
void *Correr(void  *sapo){
 int pulos = 0;
 int distanciaJaCorrida = 0;

 while (distanciaJaCorrida <= DISTANCIA_PARA_CORRER) {
    int pulo = rand() % PULO_MAXIMO;
    distanciaJaCorrida += pulo;
    pulos++;
    printf("Sapo %d pulou\n", (int) sapo);
    int descanso = rand() % DESCANSO_MAXIMO;
    sleep(descanso);
  }
     printf("Sapo %d  chegou na posicaoo %d com %d pulos\n", (int) sapo,
classificacao, pulos);
     cont++;
     classificacao++;
     pthread_exit(NULL);
}
```

Perguntas?

Exercícios!



1) Execute o programa Corrida de Sapo algumas vezes e analise os resultados sobre a ordem de chegada dos sapos. Obs: compile com a opção –lpthread

2) Usando mutex, modifique o programa Corrida de Sampo para que o problema identificado anteriormente não ocorra.

Exercícios!

3) Usando threads, escreva um programa C que implemente o problema do produtor/consumidor. O produtor deve produzir dados (números inteiros pseudo-aleatórios) a cada 1 segundo colocando-os em uma fila (buffer, implementação circular). O consumidor deve retirar dados da fila a cada 2 segundos. O tamanho máximo da fila deve ser de 8 elementos (MAXFILA) e tanto o produtor como o consumidor devem produzir/consumir 64 elementos (números inteiros de 1 a 64, por exemplo) evitando condições de corrida. Variáveis compartilhadas entre threads são simplesmente variáveis globais.

4) Modifique o programa anterior, para que haja 2 (ou mais) threads consumidor e 2 (ou mais) threads produtor. O que muda em relação ao uso do mutex e da variável de condição?