Fluxos de Execução
Um programa sequencial consiste de um único fluxo ou thread de execução , que é responsável por realizar uma certa tarefa computacional, a maioria dos programas simples possuem uma só thread e não conseguem executar duas ou mais tarefas em paralelo, mesmo em um ambiente de multiprocessamento.

Recursos vs Escalonamento
Existem duas características fundamentais que são usualmente tratadas de forma independente pelo S.O:

- Propriedade de recursos (“resource ownership”);
  Recursos alocados aos processos, e que são necessários para a sua execução (memória, arquivos, dispositivos E/S).

- Escalonamento (“scheduling / dispatching”).
  Relacionado à unidade de escalonamento do S.O que determina o fluxo de execução (trecho de código) que é executado pela CPU.

Em um sistema multi threaded:

- Processos estão associados somente à propriedade de recursos
- Threads estão associadas às unidades de execução (ou seja, threads constituem as unidades de escalonamento em sistemas multithreading).
  ,

Task Control BLock
Uma tabela de threads, denominada Task Control Block, é mantida para armazenar informações individuais de cada fluxo de execução.
Cada thread tem a si associada:

- Thread ID
- Estado dos registradores
- Endereços da pilha
- Máscara de sinais
- Prioridade
- Variáveis locais e variáveis compartilhadas com as outras threads
- Estado de execução (pronta, bloqueada, executando)

Estados de uma Thread
Uma Thread possui três estados fundamentais: executando, pronta e bloqueada.

Vantagens das Threads sobre Processos

Economia
Alocar memória e recursos para a criação de um processo é custoso, por esse motivo a criação e terminação de uma thread é mais rápida do que a criação e terminação de um processo e uma vez que threads compartilham recursos do processo ao qual elas pertencem, é mais econômico criar e fazer o escalonamento (troca de contexto) de threads do mesmo processo.

Responsividade
Transformando uma aplicação interativa em multithread pode permitir que o programa continue executando mesmo se uma parte dele se bloqueie, ou executando uma sequência grande de operações (em background), aumentando com isso a responsividade ao usuário.

Compartilhamento de recursos
Com o compartilhamento de código e dados, uma mesma aplicação pode possuir várias threads realizando atividades diferentes em cima do mesmo espaço de endereçamento.

User-level Threads - ULT
O gerenciamento das threads, incluindo o seu escalonamento, é feito no espaço de endereçamento de usuário, por meio de uma biblioteca de threads.
Muitas das chamadas ao sistema são bloqueantes e o kernel bloqueia processos – neste caso todos as threads do processo podem ser bloqueados quando uma ULT executa uma SVC.

Kernel-level Threads – KLT
O kernel pode melhor aproveitar a capacidade de multiprocessamento da máquina, escalonando as várias threads do processo em diferentes processadores.
O bloqueio de uma thread não implica no bloqueio das outras threads do processo. Assim, KLT é interessante para aplicações que bloqueiam frequentemente.

Criação de Threads: pthread_create()

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

- pthread_t *thread – ponteiro para um objeto que recebe a identificação da nova thread.
- pthread_attr_t *attr – ponteiro para um objeto que provê os atributos para a nova thread.
- start_routine – função com a qual a thread inicia a sua execução
- void *arg – argumentos inicialmente passados para a função