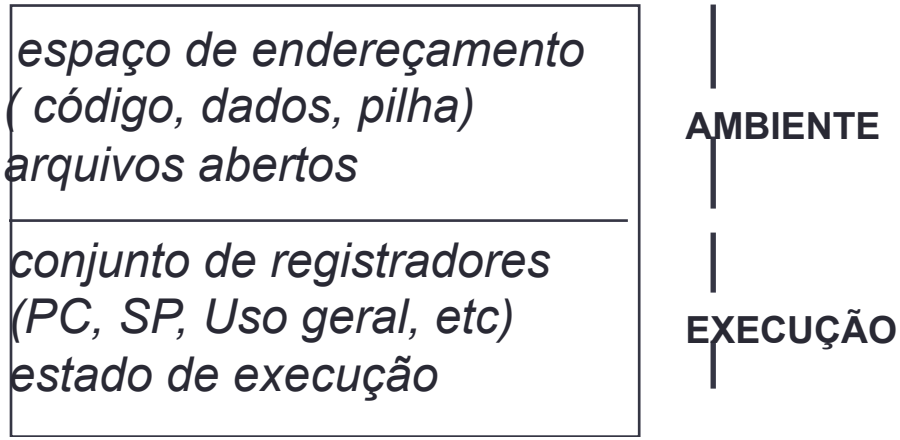
Tópico 5

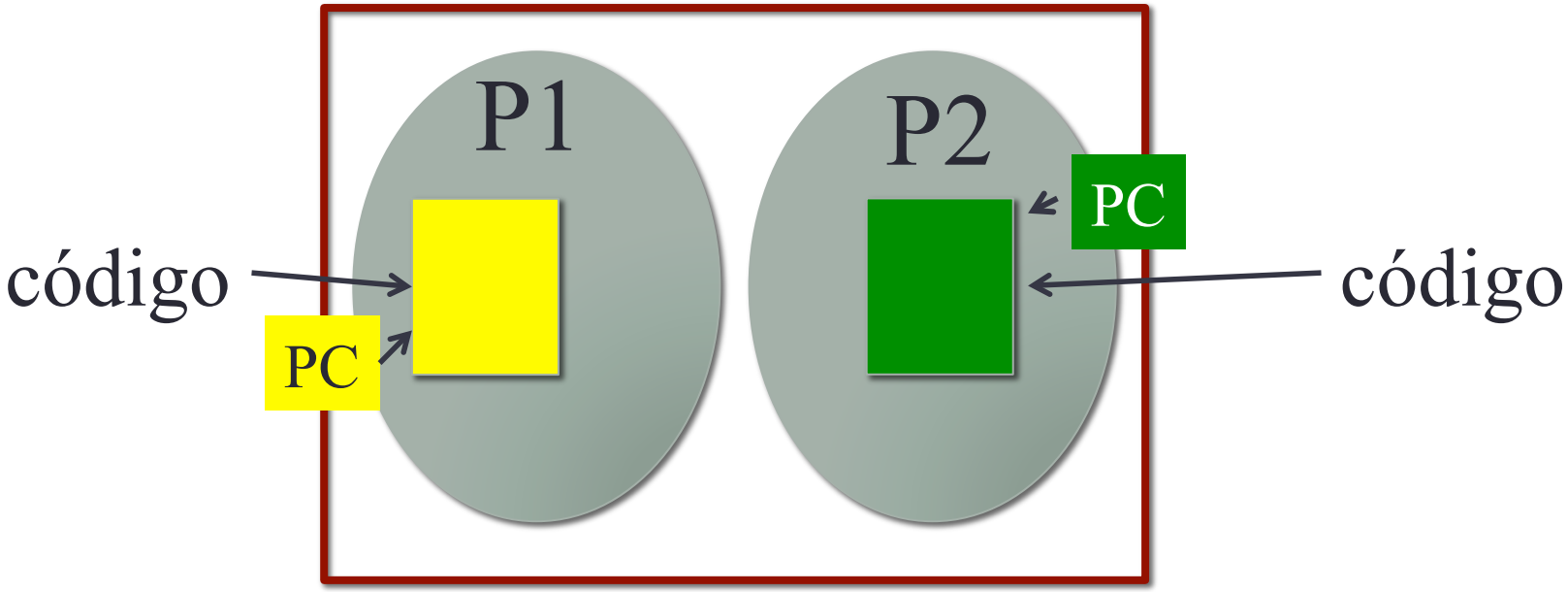
Threads

Modelos de Processos

- Classificação dos modelos de processos quanto ao custo de troca de contexto e de manutenção
  - ① "heavyweight"
  - ② "lightweight"
- Modelo de processos tradicional (heavyweight)
  - Neste caso, o processo é composto tanto pelo ambiente como pela execução.
  - Itens da tabela de processos:



Execução com Processos

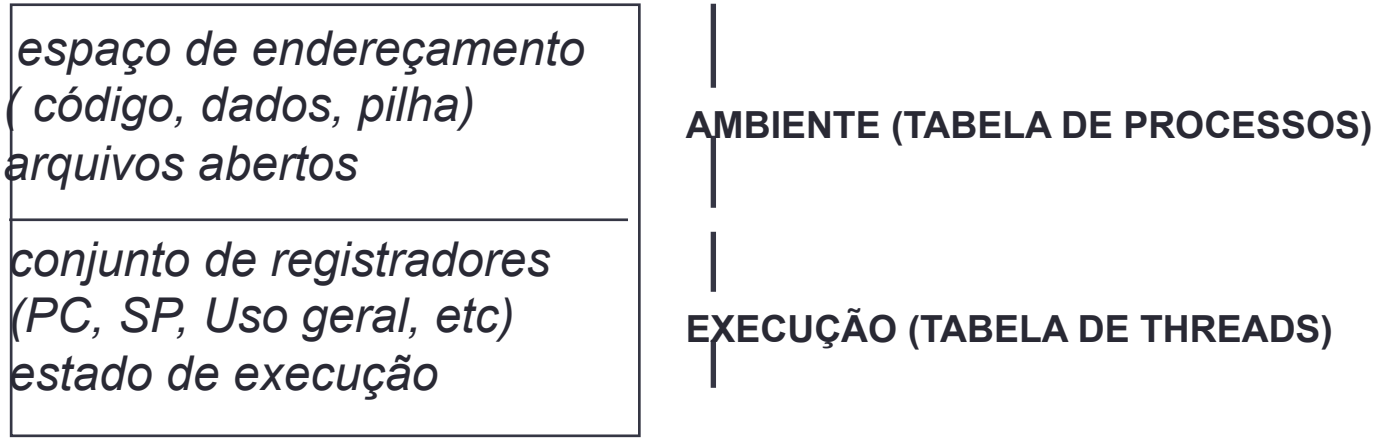


- A troca de contexto entre processos tradicionais é pesada para o sistema. Neste caso, o contexto é o ambiente e o estado de execução do processo.
- Normalmente, os processos tradicionais não compartilham memória.
- O processo tradicional é composto de uma única thread de controle

Processos Leves - Threads

- As linhas de controle múltiplas foram criadas para permitir maior concorrência na execução dos processos.
- Neste modelo, a entidade processo é dividida em duas entidades: processo (ou tarefa) e thread. O processo corresponde ao ambiente e a thread corresponde ao estado de execução. Um processo é composto por várias threads que compartilham o ambiente (memória, descritores de arquivos, etc).

Processos Leves - Threads



No modelo leve, existem duas entidades:

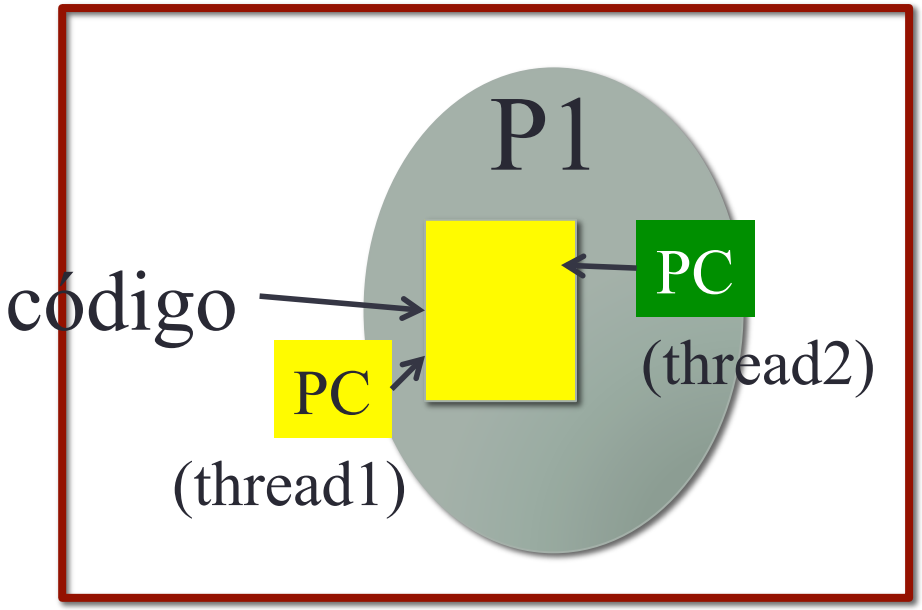
Processo: armazena informações de ambiente

Thread: armazena informações de execução

Um processo possui pelo menos uma thread.

Cada processo possui uma tabela de threads associada

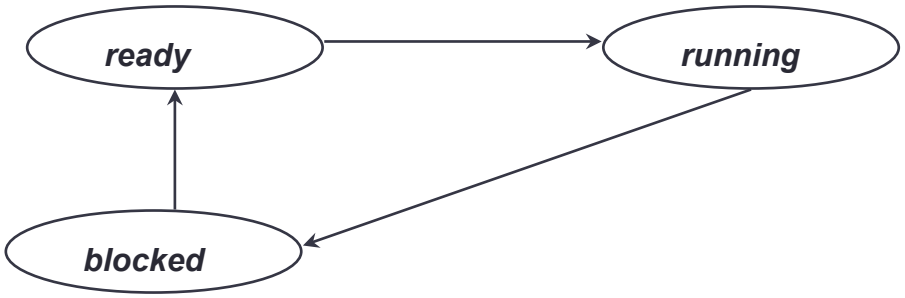
Execução com Threads



- No exemplo acima, temos 1 processo com 2 threads de controle.
- Uma thread pode se bloquear à espera de um recurso. Neste momento, uma outra thread (do mesmo processo ou não) pode se executar.
- A troca de contexto é mais leve. Se a thread t1 do processo p1 estiver rodando e entrar em execução, logo após, a thread t2 também do processo p1, a troca de contexto só diz respeito à execução. O ambiente continua o mesmo.

Estados das Threads

- A entidade que realmente se executa é a thread. O processo (ou tarefa) é só o ambiente.
- Estados de execução das threads:



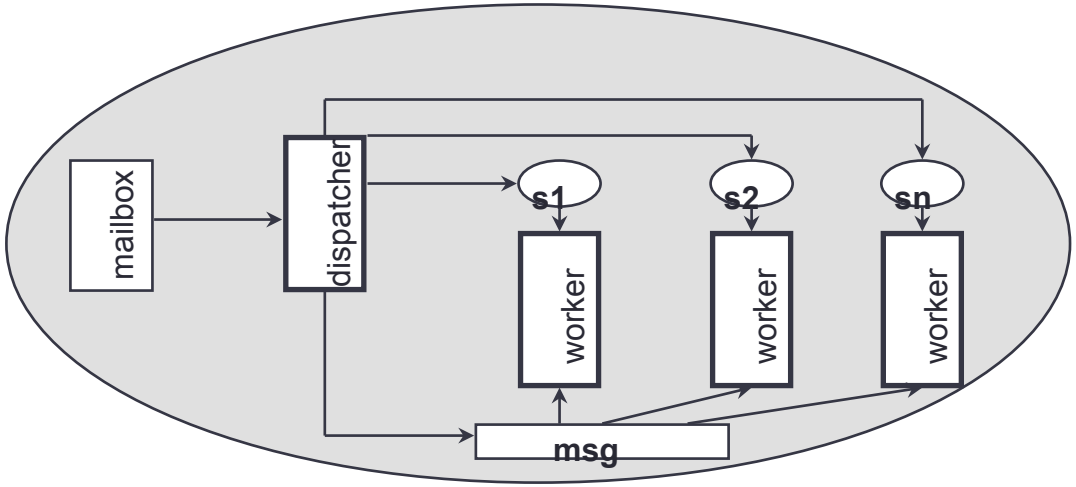
As threads compartilham as variáveis globais do programa, os descritores abertos, etc. Assim, há necessidade de mecanismos de sincronização

Modelo de Execução das Threads

- O modelo de execução determina como as threads irão se organizar para resolver um problema. É claro que esta organização é altamente dependente do problema a ser resolvido.
- Em servidores, o modelo pode ser:
  - Threads dinâmicas: uma thread é criada para tratar cada requisição
  - Threads estáticas: o número de threads é fixo
    - dispatcher/worker, team e pipeline
- Nos modelos a seguir, uma thread trata uma requisição

Modelo Dispatcher/Worker

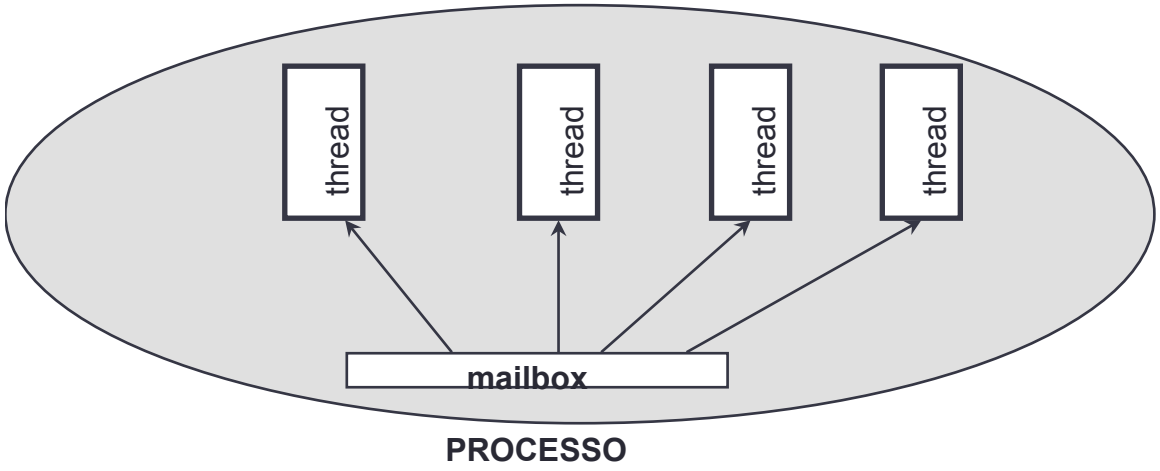
– Nesta organização, a thread dispatcher recebe todas as requisições. Ela escolhe, então, uma thread trabalhadora e a "acorda" uma thread ociosa para tratar a requisição. A thread trabalhadora executa a solicitação e, quando acaba, sinaliza o dispatcher.



- Consumo rápido de msgs
- Boa distribuição das requisições
- Flexibilidade: podemos facilmente mudar o número threads

Modelo Team

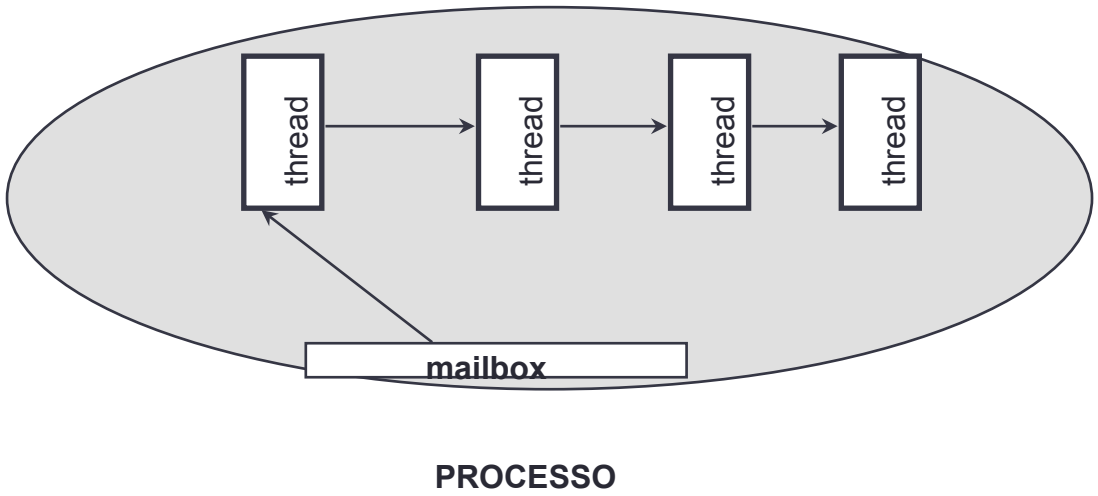
– Nesta organização, cada thread é inteiramente autônoma. Todas as threads acessam a caixa-postal, obtém requisições e as executam.



- Bom consumo de msgs
- Boa distribuição das requisições
- Flexibilidade: podemos facilmente mudar o número threads

Modelo Pipeline

– Nesta organização, cada thread tem uma tarefa específica e os dados de entrada de uma thread são produzidos pela thread anterior.



- Consumo rápido de msgs
- Aplicações produtor/ consumidor
-Menos flexível que as abordagens anteriores

Gerência de Processos Distribuídos
Comparação entre os modelos de processos

- **Modelos:**
  - Processos tradicionais: Não há concorrência no interior do processo. Existem chamadas de sistema bloqueadas
  - Processos + threads: Há concorrência no interior do processo. Existem chamadas de sistema bloqueadas
  - Máquina de estados finitos: Neste modelo, são guardados os estados parciais de execução de solicitações. Não há chamadas de sistema bloqueadas.

- Os sistemas que utilizam processos tradicionais são programados de maneira relativamente simples, porém não permitem uma grande concorrência. Os sistemas à máquina de estados permitem concorrência no tratamento de requisições porém sua programação é complexa. O modelo de threads foi proposto para proporcionar concorrência e facilidade de programação ao mesmo tempo.

Gerência de Processos Distribuídos
Aspectos no projeto das threads

❶ **Tipos de threads**:

- Estáticas: o número de linhas de controle que comporão um processo é determinado em tempo de compilação. A execução começa com n threads.
- Dinâmicas: as threads são criadas e destruídas dinamicamente, ao longo da execução. As threads são criadas através de primitivas. A execução começa com uma thread apontando para o início da rotina main.

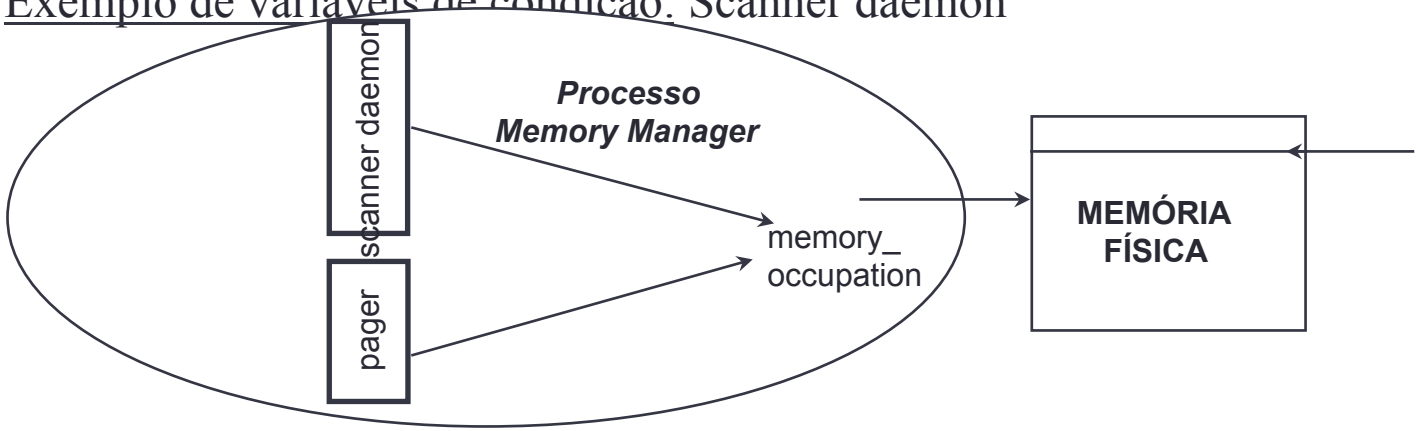
  ➔ técnica mais utilizada

❷ **Sincronização:**

- Como as threads compartilham memória, elas necessitam de mecanismos de sincronização. A maioria dos pacotes de threads oferece dois mecanismos: variáveis mutex e variáveis de condição.
  - variáveis mutex: semáforo binário
    - dois estados: fechado ou aberto
    - operações: lock, unlock, trylock
  - variáveis de condição: uma variável de condição é sempre associada a um mutex (semáforo).
    - semântica: obtem o semáforo, testa a condição. Se a condição for verdadeira, continua. Senão, solta o semáforo e fica esperando na

Gerência de Processos Distribuídos
Aspectos no projeto das threads

- Exemplo de variáveis de condição: Scanner daemon



- Scanner (paginador):

```
pthread_mutex_lock(v);
while (memory_occupation < M_BOUND)
   pthread_cond_wait(
      memory_full, v);
livres = libera_paginas();
memory_occupation = memory_occupation - livres;
pthread_mutex_unlock(v);
```

- Pager (alocador de páginas):

Gerência de Processos Distribuídos
Aspectos no projeto das threads

❷ **Tratamento de variáveis globais:**

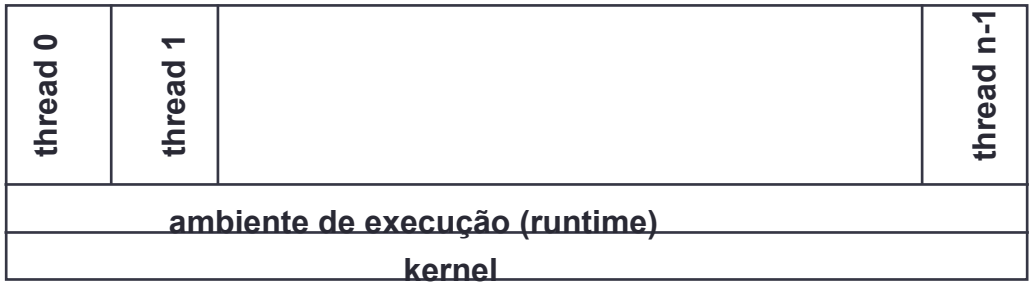
- Como as variáveis globais são acessadas agora por diversas threads, deve-se pensar em uma maneira de se garantir a não interferência entre as mesmas.
- Soluções:
  - *Não permitir variáveis globais*. Não é boa porque vários softwares, inclusive o Unix, utilizam variáveis globais
  - *Criar um conjunto de variáveis globais para cada thread* (variáveis globais privadas) que só são vistas pela thread que as criou e pelos outros processos do sistema. Utilizar um conjunto de primitivas: create_global, read_global, set_global.
    - Não é uma boa solução pois aumenta a complexidade da programação e a complexidade da semântica das threads,
  - *Proteger o acesso às variáveis globais com mecanismos de sincronização*. Não é uma solução genérica, pois algumas variáveis globais são setadas pelo próprio sistema (e. g. errno).
  - *Colocar as chamadas ao sistema que setam as variáveis globais em uma única thread*. Não é uma boa solução pois implica a perda de concorrência.

Implementação das threads

- Em que nível implementar as threads?
  - ❶ Implementar o modelo de processos e threads à nível de sistema operacional, criando as abstrações de processo e de thread.
  - ❷ Manter o modelo de processos heavyweight e simular as múltiplas threads através de biblioteca.
- Implementação de threads à nível de sistema operacional
  - O Kernel do sistema operacional é responsável pela criação das threads, seu escalonamento e seu término.
  - Existência de uma tabela de threads no kernel, que contem os dados inerentes às threads.
  - As threads são manipuladas através de chamadas ao sistema.
  - Quando uma thread é bloqueada, o kernel escolhe uma outra thread para rodar (do mesmo processo ou de processos diferentes).
  - Também são chamadas "heavyweight threads" porque o tempo envolvido na troca de contexto é considerável, apesar de menor do que o tempo de troca de contexto entre processos.
  - O sistema operacional distribuído Mach implementa threads a nível de sistema operacional.
  - ✓ Infelizmente, as threads implementadas à nível de sistema operacional

Gerência de Processos Distribuídos
Implementação das threads

- Implementação de threads à nível de biblioteca
  - As threads são simuladas no espaço do usuário. O sistema operacional continua tratando com processos comuns.



  - O ambiente de execução é o responsável pelo gerenciamento das linhas de controle.
  - Existência de uma tabela de threads no ambiente de execução, que contem os dados inerentes às threads.
  - As threads são manipuladas através de chamadas à funções.
  - Geralmente o escalonador do runtime é não-preemptivo.
  - Sempre que uma linha de controle t1 for suspender sua execução ela chama um procedimento do ambiente de execução. Neste momento, o ambiente de execução salva os registradores e a pilha de t1 na tabela de threads, escolhe

Gerência de Processos Distribuídos
Implementação das threads a nível de usuário

- Troca de contexto extremamente rápida.
- Cada processo pode ter o seu próprio algoritmo de escalonamento.
- Problema: As chamadas de sistema bloqueadas causam o bloqueio de todas as threads do processo! Por essa razão, elas não podem ser feitas diretamente pelas threads.
- Solução: Colocar uma capa (jacket) sobre as chamadas blocantes ao sistema. Com essa técnica, as chamadas blocantes são "mascaradas" pela biblioteca de threads, que faz um teste de bloqueio. Se a chamada ao sistema resultar em bloqueio, ela só é feita se não houver nenhuma thread esperando. Se houver alguma thread esperando para se executar, a thread anterior é bloqueada e a nova thread entra em execução. jacket: inserção de código de verificação às chamadas. Só funciona com bibliotecas standard.

Gerência de Processos Distribuídos
Implementação das threads

- Análise dos tipos de implementação
  - "As threads foram introduzidas para dar mais concorrência à execução das aplicações a um custo relativamente baixo".
  - Implementação à nível de kernel: alta concorrência potencial porém o custo de troca de contexto entre threads continua alto.
  - Implementação à nível de usuário: pouca concorrência devido ao bloqueio de todas as threads do processo quando uma delas faz operações de I/O O custo da troca de contexto é extremamente baixo.
- Conclusão: Ainda não existe uma boa implementação de threads.
- Situação atual: A maioria dos sistemas implementa threads a nível de usuário, por questões de comodidade (não é necessário alterar o SO) ou desempenho. Os poucos sistemas que implementam threads a nível de sistema operacional (e.g. Mach) oferecem também uma interface mais amigável através de um conjunto de bibliotecas C (C threads) e, estas sim, fazem as chamadas ao sistema.

Gerência de Processos Distribuídos
Exemplo de pacote de threads

- **Pacote POSIX threads (pthreads)**
  - Desenvolvido pela OSF
  - Faz parte do Distributed Computing Environment (DCE)
  - Chamadas de controle de threads:
    - *pthread_create* - cria uma thread
    - *pthread_exit* - termina a execução da thread
    - *pthread_join* - espera que a thread-filha termine
    - *pthread_detach* - informa à thread-mãe que esta não deve esperar por seu término
    - *pthread_yield* - libera o processador
  - Chamadas de sincronização
    - *pthread_mutex_init* - cria um mutex
    - *pthread_mutex_destroy*
    - *mutex_lock* - obtem lock blocante
    - *mutex_unlock* - libera o lock
    - *mutex_trylock* - testa o estado do lock
    - *cond_init* - cria uma variável de condição

Gerência de Processos Distribuídos
Exemplo de pacote de threads

- **Pacote POSIX threads (pthreads)**
  - Chamadas de tratamento de variáveis globais privadas:
    - *pthread_keycreate* - cria uma variável global privada
    - *pthread_setspecific* - atribui um valor à variável
    - *pthread_getspecific* - lê valor da variável
  - Chamadas de escalonamento:
    - *pthread_setscheduler* - atribui uma determinada política de escalonamento à thread: FIFO, round-robin, preempção, etc
    - *pthread_getscheduler* - obtem o algoritmo corrente de escalonamento
    - *pthread_setprio* - atribui um valor à prioridade de escalonamento da thread
    - *pthread_getprio* - lê a prioridade de escalonamento da thread