Sistemas Operacionais: Threads

Prof. Dr. Rafael Lopes Gomes
Universidade Estadual do Ceará (UECE)

Agenda

- Processos
- **Threads**
- Comunicação entre processos
- Escalonamento de Processos

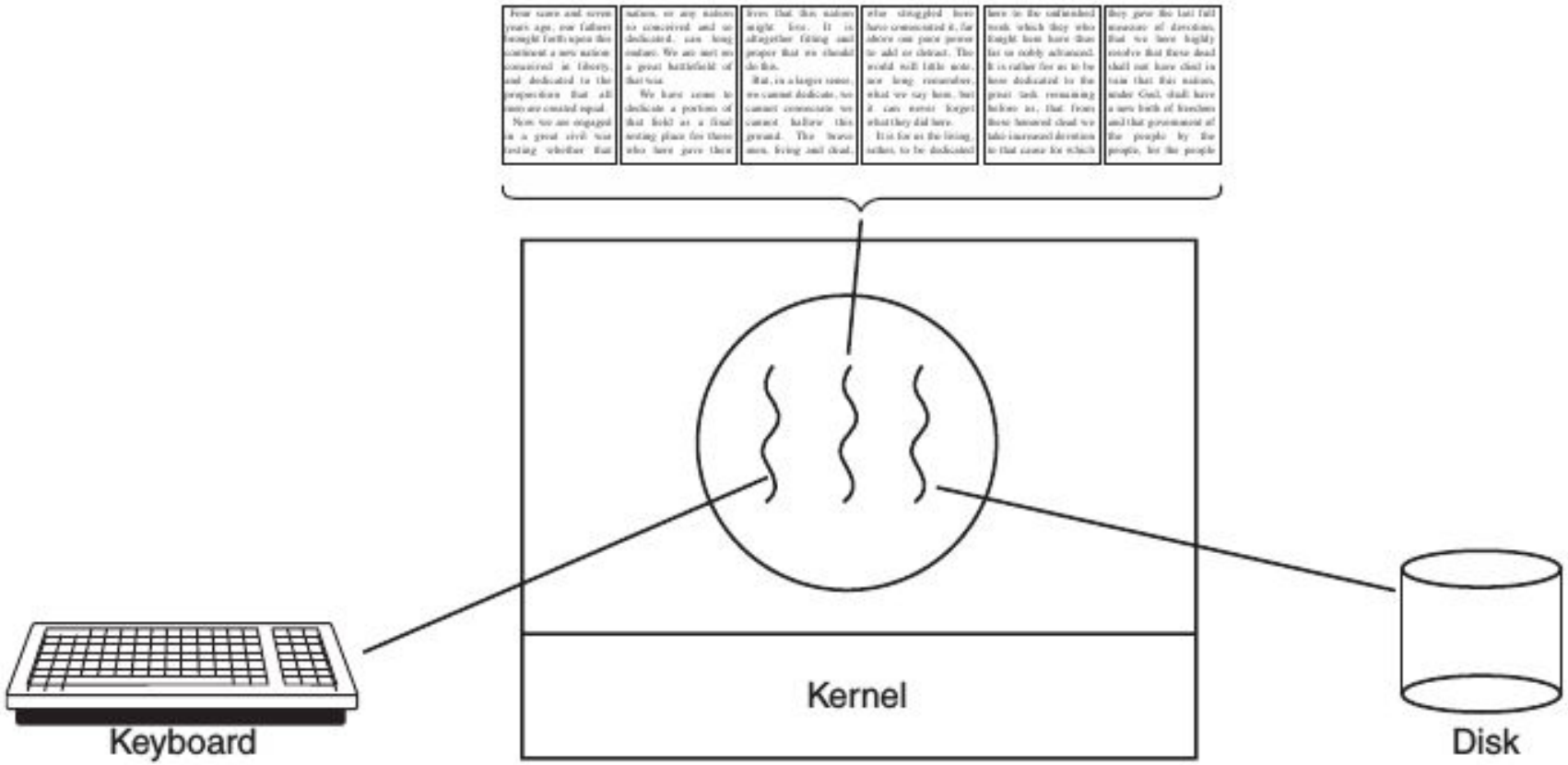
Threads

- Definição: é uma linha de execução dentro do processo;
- SO tradicionais: Cada processo tem um espaço de endereçamento e uma única thread de controle/execução.
- Múltiplas threads em um mesmo espaço de endereçamento:
  - Executam quase em paralelo;
  - Como se fossem processos separados.

Threads: Motivação

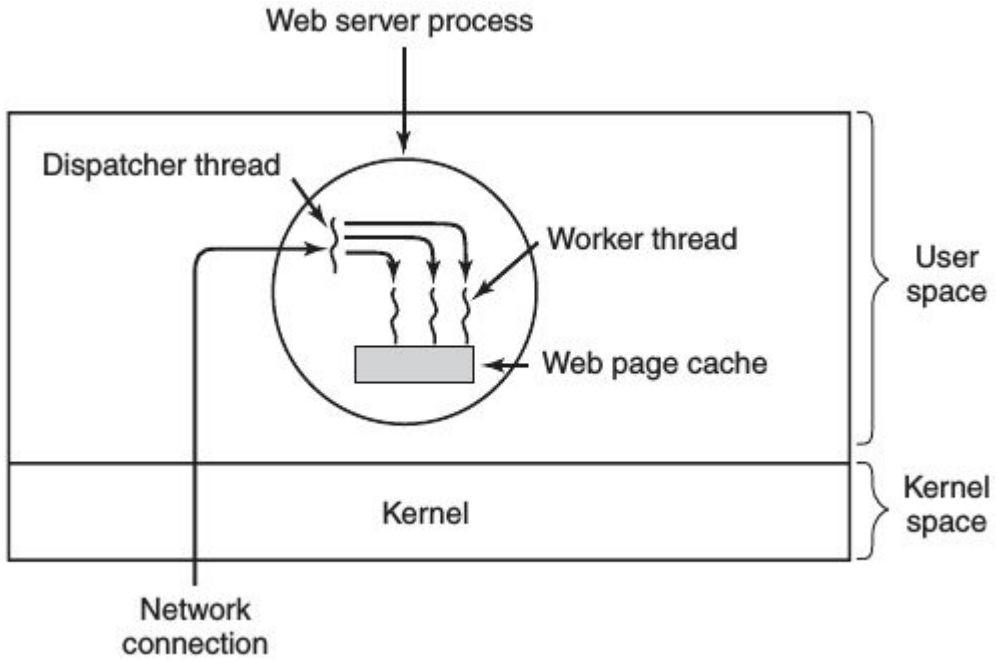
- Muitas aplicações com múltiplas atividades estão concorrendo simultaneamente, onde algumas podem bloquear de tempos em tempos:
  - Decompor essas aplicações em várias threads que executam quase em paralelo;
  - Compartilham espaço de endereçamento;
- Threads são mais leves que processos:
  - Mais fácil criar e destruir;
- Desempenho: Sobreposição de atividades;
- Maior capacidade de paralelismo.

Threads: Exemplo



Threads: Servidor Web

- Uma thread para receber as requisições (despachante);
- Threads operário examinam e executam as operações
  - Thread adormecida enquanto ociosa (não disputa CPU);
  - Thread criada para atender;



Threads: Servidor Web

Despachante

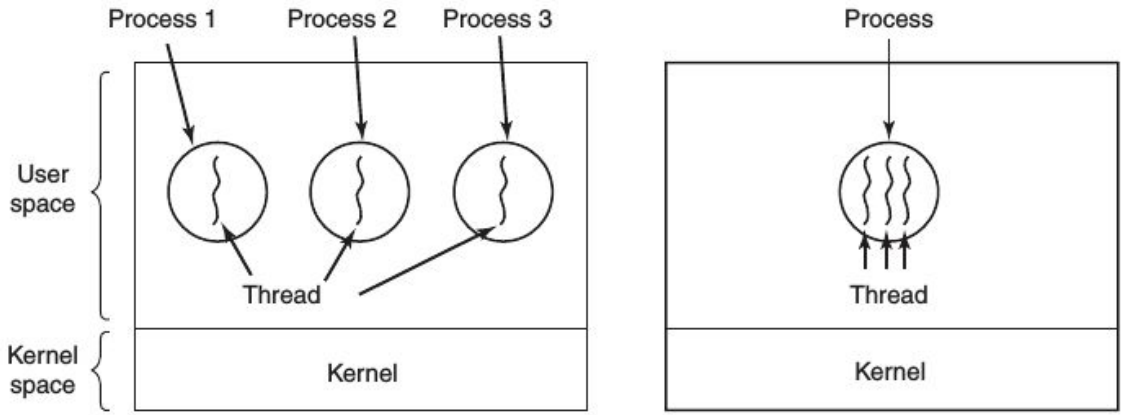
```
while (TRUE) {
    get_next_request(&buf);
    handoff_work(&buf);
}
```

Operário

```
while (TRUE) {
    wait_for_work(&buf)
    look_for_page_in_cache(&buf, &page);
    if (page_not_in_cache(&page))
        read_page_from_disk(&buf, &page);
    return_page(&page);
}
```

Threads

- Modelo de processo:
  - Execução (Thread).
  - Agrupamento de recursos;
- Múltiplas threads:
  - Diversas execuções no mesmo ambiente;
  - Independência de execução.
  - Multithread: múltiplas threads em um mesmo processo
    - CPU chaveia entre as threads, dando a ilusão de paralelismo.



Threads e Processos

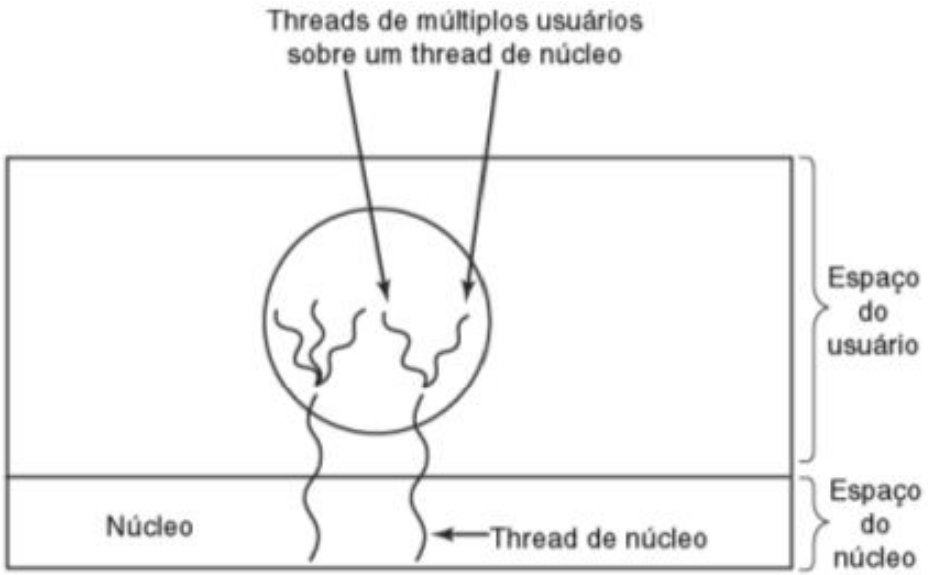
- Threads de um mesmo processo compartilham o mesmo espaço de endereçamento;
  - Herdam arquivos abertos, temporizadores, sinais, semáforos, variáveis globais, etc.
- Não existe proteção entre threads de um mesmo processo;
  - Uma thread pode ler e escrever em outra;
- Transições de estado nas threads são as mesmas dos processos
  - Bloqueado, em execução e Pronto.

Threads

- Normalmente, os processos começam com uma única thread que tem a capacidade de criar outras
  - Pthread_create
- Uma thread pode concluir a execução explicitamente
  - Pthread_exit
- Uma thread pode esperar a execução de uma outra, bloqueando a thread até o término da outra
  - Pthread_join

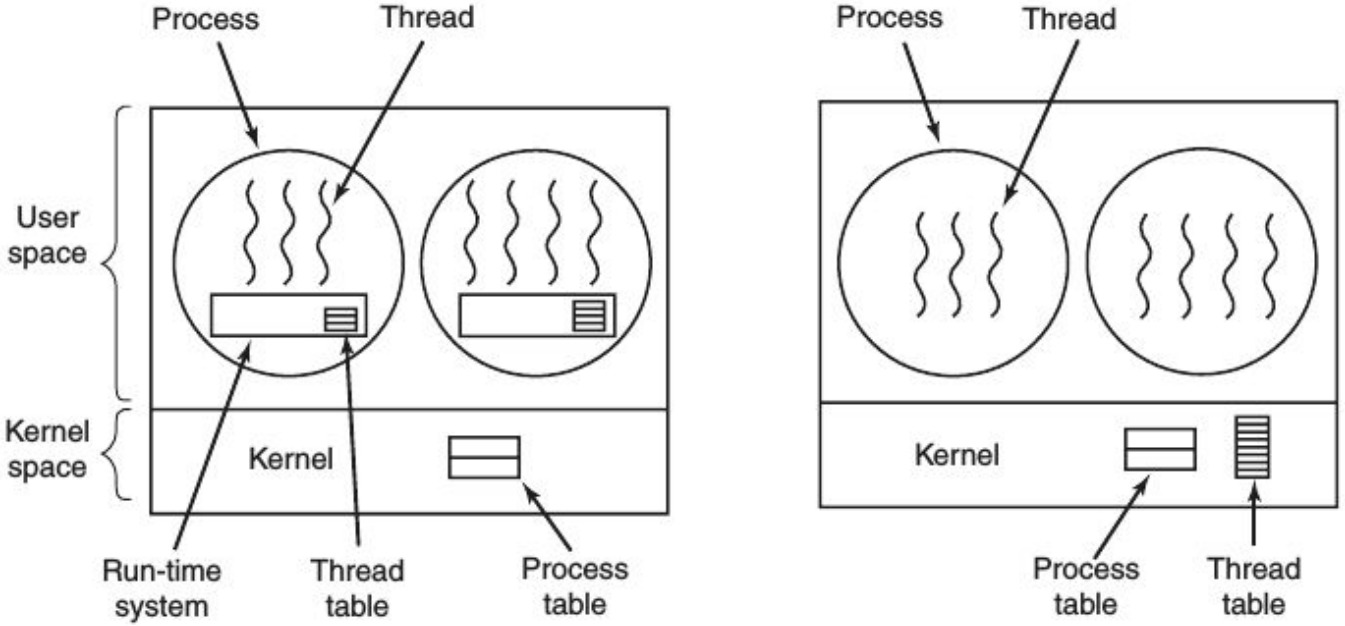
Threads: Implementação

- Modelos de Implementação:
  - Espaço de usuário (como uma biblioteca)
    - Ex: DOS
  - Espaço de núcleo (chamadas)
    - Ex: Windows e Linux
  - Híbrido ou Multiplexação
    - M threads de um processo são tratadas como N threads pelo Kernel



Threads: Implementação

- Espaço de usuário
  - Threads são implementadas por bibliotecas;
  - O núcleo acha que é um processo monothread;
  - Cada processo tem sua própria tabela de threads



Threads: Espaço de usuário

- Vantagens
  - Troca de contexto é rápida
  - Escalonamento particular
  - Melhor desempenho e escalabilidade
- Desvantagens
  - Bloqueio de threads
  - Problemas de utilização do processador

Threads: Espaço de Núcleo

- Para criar ou destruir uma thread é feita uma chamada de sistema, atualizando a tabela de threads do núcleo;
- Evita o problema de bloqueio
- Reciclagem de threads:
  - Uma thread destruída é marcada como não executável
  - Estrutura não é afetada
  - Nova thread reativa threads marcadas (evita sobrecarga)
- Desvantagem: custo da chamada de sistema é maior.

Threads: create5.c

```
#define N_THR 10
void* f_thread(void *v) {
  int thr_id;
  sleep(1);
  thr_id = *(int*) v;
  printf("Thread %d.\n", thr_id);
  return NULL;
}
int main() {
  pthread_t thr[N_THR];
  int i, *p_id;
  for (i = 0; i < N_THR; i++) {
    p_id = (int*) malloc(sizeof(int));
    *p_id = i;
    pthread_create(&thr[i], NULL, f_thread, (void*) p_id);
  }
  for (i = 0; i < N_THR; i++)
    pthread_join(thr[i], NULL);
  return 0;
}
```

Threads: create_join.c

```
#define N_THR 10
void* f_thread(void *v) {
  int thr_id = (int) v;
  pthread_t thr;
  if (thr_id + 1 < N_THR)
    pthread_create(&thr, NULL, f_thread, (void*) thr_id + 1);
  if (thr_id + 1 < N_THR)
    pthread_join(thr, NULL);
  printf("Thread %d iniciou o seu trabalho.\n", thr_id);
  sleep(random() % 3);
  printf("Thread %d terminou o seu trabalho.\n", thr_id);
  return NULL;
}
int main() {
  pthread_t thr;
  pthread_create(&thr, NULL, f_thread, (void*) 0);
 sleep(random() % 3);
  pthread_join(thr, NULL);
  return 0;
}
```

Threads: pthread_return.c

```
void* f_thread(void *v) {
  int r;
  printf("Valor a ser retornado: ");
  scanf("%d", &r);
  return (void*) r;
}

int main() {
  pthread_t thr;
  int retorno;
  pthread_create(&thr, NULL, f_thread, NULL);
  pthread_join(thr, (void **) &retorno);
  if (retorno == 0)
      printf("Thread encerrou normalmente.\n");
  else
      printf("Thread encerrou com valor: %d\n", retorno);
  return 0;
}
```

Threads: pthread_exit0.c

```
void g(int r) {
  pthread_exit((void*) r);
}
void* f_thread(void *v) {
  int r;
  printf("Valor a ser retornado: ");
  scanf("%d", &r);
  g(r);
  return (void*) r+1; /* Nunca executado */
}
int main() {
  pthread_t thr;
  int retorno;
  pthread_create(&thr, NULL, f_thread, NULL);
  pthread_join(thr, (void **) &retorno);
  if (retorno == 0)
     printf("Thread encerrou normalmente.\n");
  else
     printf("Thread encerrou com valor: %d\n", retorno);
  return 0;
}
```

FIM